DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Real-time forecasting of dietary habits and user health using Federated Learning with privacy guarantees

SONIA-FLORINA HORCHIDAN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Real-time forecasting of dietary habits and user health using Federated Learning with privacy guarantees

SONIA-FLORINA HORCHIDAN

Master's programme in Software Engineering of Distributed Systems
Date: August 21, 2020
Supervisor: Zainab Abbas and Sana Imtiaz
Examiner: Vladimir Vlassov
School of Electrical Engineering and Computer Science
Swedish title: Realtidsprognoser av matvanor och användarhälsa med hjälp av Federerade Lärande med integritetsgarantier

Abstract

Modern health self-monitoring devices and applications, such as Fitbit and MyFitnessPal, empower users to take concrete actions and set fitness and lifestyle goals based on their recorded trends and statistics. Predicting such trends is beneficial in the road of achieving long-time targets, as the individuals can adjust their diets and habits at any point to guarantee success. The design and implementation of such a system, which also respects user privacy, is the main objective of our work.

This application is modelled as a time-series forecasting problem. Given the historical data of users, we aim to predict their eating and lifestyle habits in real-time. We apply the federated learning paradigm to our use-case because of the highly-distributed nature of our data and the privacy concerns of such sensitive recorded information. However, federated learning from heterogeneous sequences of data can be challenging, as even state-of-the-art machine learning techniques for time-series forecasting can encounter difficulties when learning from very irregular data sequences. Specifically, in the proposed healthcare scenario, the machine learning algorithms might fail to cater to users with unique dietary patterns.

In this work, we implement a two-step streaming clustering mechanism and group clients that exhibit similar eating and fitness behaviours. The conducted experiments prove that learning federatively in this context can achieve very high prediction accuracy, as our predictions are no more than 0.025% far from the ground truth value with respect to the range of each feature. Training separate models for each group of users is shown to be beneficial, especially in terms of the training time, but it is highly dependent on the parameters used for the models and the training process. Our experiments conclude that the configuration used for the general federated model cannot be applied to the clusters of data. However, a decrease in prediction error of more than 45% can be achieved, given the parameters are optimized for each case.

Lastly, this work tackles the problem of data privacy by applying state-of-the-art differential privacy techniques. Our empirical study shows that noising the gradients sent to the server is unsuitable for small datasets and cancels out the benefits obtained by prior users' clustering. On the other hand, noising the training data achieves remarkable results, obtaining a differential privacy level corresponding to an epsilon value of 0.1 with an increase in the observed mean absolute error by a factor of only 0.21.

Keywords: Federated Learning, Time Series Forecasting, Clustering, Pattern Matching, Real-time Data Processing, Differential Privacy, Data Privacy.

Sammanfattning

Moderna apparater och applikationer för självövervakning av hälsa, som Fitbit och MyFitnessPal, ger användarna möjlighet att vidta konkreta åtgärder och sätta fitness- och livsstilsmål baserat på deras dokumenterade trender och statistik. Att förutsäga sådana trender är fördelaktigt för att uppnå långtidsmål, eftersom individerna kan anpassa sina dieter och vanor när som helst för att garantera framgång.

Utformningen och implementeringen av ett sådant system, som dessutom respekterar användarnas integritet, är huvudmålet för vårt arbete. Denna applikation är modellerad som ett tidsserieprognosproblem. Med avseende på användarnas historiska data är målet att förutsäga deras matvanor och livsstilsvanor i realtid. Vi tillämpar det federerade inlärningsparadigmet på vårt användningsfall på grund av den mycket distribuerade karaktären av vår data och integritetsproblemen för sådan känslig bokförd information. Federerade lärande från heterogena datasekvenser kan emellertid vara utmanande, eftersom även de modernaste maskininlärningstekniker för tidsserieprognoser kan stöta på svårigheter när de lär sig från mycket oregelbundna datasekvenser. Specifikt i det föreslagna sjukvårdsscenariot kan maskininlärningsalgoritmerna misslyckas med att förse användare med unika dietmönster.

I detta arbete implementerar vi en tvåstegsströmmande klustermekanism och grupperar användare som uppvisar liknande ät- och fitnessbeteenden. De genomförda experimenten visar att federerade lärande i detta sammanhang kan uppnå mycket hög nogrannhet i förutsägelse, eftersom våra förutsägelser inte är mer än 0,025% ifrån det sanna värdet med avseende på intervallet för varje funktion. Träning av separata modeller för varje grupp användare visar sig vara fördelaktigt, särskilt gällande träningstiden, men det är mycket beroende av parametrarna som används för modellerna och träningsprocessen. Våra experiment drar slutsatsen att konfigurationen som används för den allmänna federerade modellen inte kan tillämpas på dataklusterna. Dock kan en minskning av förutsägelsefel på mer än 45% uppnås, givet att parametrarna är optimerade för varje fall.

Slutligen hanteras problemet med datasekretess genom att tillämpa bästa tillgängliga differentiell integritetsteknik. Vår empiriska studie visar att addera brus till gradienter som skickas till servern är olämpliga för liten data och avbryter fördelarna med tidigare användares kluster. Däremot, genom att addera brus till träningsdata uppnås anmärkningsvärda resultat. En differentierad integritetsnivå motsvarande ett epsilonvärde på 0,1 med en ökning av det observerade genomsnittliga absoluta felet med en faktor på endast 0,21 erhölls.

Nyckelord: Federerade Lärande, Tidsseriesprognos, Klustergruppering, Mönstermatchning, Realtidshantering av data, Differentialintegritet, Dataintegritet

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter we present the motivation and problem statement of this study. Then, we introduce the requirements of the problem, the objectives we wish to accomplish, and the main contributions. Finally, we discuss the ethics and sustainability aspects of this work. Also, we give an overview of the structure of this thesis.

1.1 Motivation and problem statement

Health tracking devices have seen a massive growth in popularity in the last 30 years. Although the fitness trackers first appeared in the 1960s, it was only in 1977 when the introduction of Polar watches ignited people's desire and interest in self-monitoring their health. The innovation brought by the Polar watches was the addition of wireless heart rate monitors [1]. Later, 3D accelerometers have been added to mobile phones to accurately track the physical activity of the users. Today, modern health trackers, such as the ones developed by Fitbit [2], come with multiple sensors capable to record complex health metrics, like the caloric intake, sleep quality, or activity level of the wearer. In addition, health-tracking mobile applications, such as MyFitnessPal [3], are freely available for users and, in some cases, can be paired with the wearable devices to give a more intricate glimpse into one's health.

Diet and health self-monitoring are powerful tools when it comes to achieving fitness and well-being goals because it can offer an objective view of the progress made over time. It can also provide meaningful insights into the trends and customs of each individual, raising awareness of perhaps undesirable habits. Our work aims to make use of such recorded information.

A very valuable application is the prediction of health data. For instance, an athlete aiming to compete in a national competition might want to set dietary goals, such as aiming to consume a certain amount of calories by the end of the month. In this case, it might be useful to observe the predicted evolution of the caloric intake of the following weeks based on historic data. This might eliminate the individual's doubts on whether the chosen diet is appropriate or if adjustments have to be made to successfully reach the target. Such a system constitutes the focus of our work.

The nature of our problem raises a unique opportunity: making use of various likenesses that can appear between different individuals. For example, people coming from a common background might show similarities in their diets, while people working certain jobs might show similar sleep schedules. A forecasting system might utilize this information to its advantage and improve its performance.

The data that is collected from health trackers is continuous, as long as the user chooses to wear the device. Any system that learns from such data and aims to provide meaningful results should account for the streaming nature of the data. In this case, using neural networks requires online learning, where the learning model adapts according to the changes in the data distribution.

Lastly, privacy concerns must be taken into consideration when it comes to healthcare data. In light of the General Data Protection Regulation (GDPR) in the European Union, the systems that operate on sensitive data should provide strict data privacy and protection guarantees. In the case of data collected from wearable devices, the system might unintentionally leak information such as racial origin, religious beliefs, and physical or mental health, if no privacy preservation method is applied. However, introducing such methods might lead to an overhead in terms of performance of the overall system as well as affect the quality of the model.

Our work aims to design and implement an end-to-end forecasting solution for dietary habits and user health with privacy control over sensitive data. Given this goal, we formulate the following problem statement:

Given a stream of data recorded by fitness tracking devices, a very accurate solution can be built to forecast user data. In such a system, users can be clustered based on their similarities to provide personalized predictions. Adding privacy preservation methods in such a system may decrease the accuracy of the forecast, but a good trade-off can be found between the accuracy and the level of added privacy preservation.

1.2 Requirements

The forecasting solution that will be built in this work comes with a series of requirements. In this subsection, we briefly mention all of these requirements, as they will motivate the objectives of our work.

Firstly, it is well-known fact that wearable fitness-tracking devices have experienced a massive increase in popularity in the past decade. The best example can be observed in the case of Fitbit devices. In 2019, approximately 30 million users were wearing a Fitbit watch [4]. All of these users are sending data per second. Thus, the designed forecasting solution has to be able to account for massive amounts of streaming data coming from users in real-time.

Additionally, the users might not be always available. The wearable devices provided by fitness-tracking companies are usually accompanied by a mobile application, which is designed to provide better visualization of the recorded data and to also communicate with the server when necessary. However, the mobile phones might be offline, might have a poor network connection or low levels of battery. In these cases, communicating with a server might not be possible. This aspect should be kept in mind when designing a forecasting system.

The system must be able to provide accurate forecasts for all its clients. It is expected that some individuals might have uncommon lifestyles and eating patterns. The system should still provide accurate predictions for these users, despite their particularities.

Lastly, as the system handles very sensitive data, the system should offer a solution to guarantee client confidentiality.

1.3 Objectives

The objectives of this thesis are as follows:

1. Given health care data streams, forecast the dietary habits and health data of the clients using federated learning.
2. Group similar users into clusters and evaluate whether training individual models for each group brings benefits in terms of accuracy and performance.
3. Evaluate the performance of the proposed clustering mechanism on the streaming data.

4. Apply differential privacy techniques on the proposed framework and compare the accuracy of the prediction on various privacy levels.

1.4 Contributions

The main contributions of this work can be summarized as follows:

1. We proposed a novel time-series clustering technique that can handle multi-variate time series. To the best of our knowledge, the problem of clustering multi-variate time series is complex and the solutions usually rely on computationally heavy operations. Moreover, the literature study shows that the already existing methods are usually not generalized and require domain specific information.
2. We implemented the proposed clustering mechanism in Apache Flink. Our proposed pipeline is domain independent and can be successfully used to cluster any streaming data of this shape.
3. We designed and implemented an end-to-end pipeline for time series forecasting on health data streams in a federated learning environment.
4. We compared the baseline model (i.e. the model trained on the whole dataset) with the models trained on clusters of users obtained by our clustering mechanism in terms of accuracy and training time.
5. We integrated state-of-the-art differential privacy approaches into our pipeline and did an extensive study of their effects in terms of accuracy for both the baseline model and the clustered models.

1.5 Ethics and sustainability aspects

As humanity is moving towards a fully digitized era, an increasing amount of data is being stored and used to provide intelligent, intuitive, and personalized services. Yet, improper usage of information can lead to harmful outcomes. The healthcare systems are the ones at utmost risk, as attacks on the database of clients' records can leak very sensitive information about the individuals.

Classical machine learning approaches require an increasing amount of personal data stored and collected to guarantee the success of the algorithm. However, a new learning paradigm emerged in recent years, which is set to solve the centralized data collection issue. Federated learning makes it possible to train powerful models, but by bypassing the need for a central entity that stores the data. As each client stores their confidential information on their own device, the risk of having the service providers leak sensitive data is highly reduced.

Another major advantage of federated learning comes in terms of sustainability. Federated learning can drastically reduce the communication cost between the local devices and the central authority, as only small updates have to be sent to the cloud. Moreover, storing the model at client level achieves lower latency, which in turn can remarkably improve the performance of applications that handle real-time data analysis and processing.

Next, we discuss the sustainability aspects of improving the performance of the federated model by taking advantage of similarities between data entities. In many applications, a data exploration step could be troublesome given the scale of data. This is especially true for federated learning applications, where the data is highly distributed. Our proposed clustering method aims to provide meaningful insights into the structure of the data and feed it to the federated learning process. Hence, we alleviate the need of dedicated experts and build a framework that is domain-independent.

As for applying privacy preservation techniques, the ethics gains are evident. Due to the sensitive nature of our data, we add another layer of privacy preserving methods on top of the guarantees offered by federated learning. This is achieved by using differential privacy, which, in our case, is employed as a practice aiming to ensure no private data can be leaked by intercepting the updates sent from the clients to the central entity.

1.6 Outline of the thesis

The rest of this thesis is organised as follows. In Chapter 2, we introduce the concepts required to comprehend the contributions of this work and the related studies found in the literature. Chapter 3 presents the design and implementation decisions made to construct our system. Chapter 4 shows the evaluation methodology chosen for our approach. Chapter 5 presents the results of our study on four datasets. In chapter 6, we discuss the conclusions that were drawn from our case studies in regards to our research questions and propose potential future work.

Chapter 2

Background

This chapter briefly introduces the concepts needed to comprehend the contributions of our research. The proposed work aims to predict the dietary habits and user health of individuals in a real-time application, while taking into consideration the similarities that might exist among them. Firstly, we address the problem of time-series clustering, followed by a summary of the main concepts in machine learning and deep learning, with a focus on the most popular network architectures used for time-series forecasting. Then, we outline the key concepts of federated learning and move towards the topic of differential privacy. We describe the framework used to implement our pipeline, Apache Flink. Lastly, we summarize a list of studies that tackle similar problems.

2.1 Time-series data

Time-series are sequences of observations that are, usually, ordered by some temporal information. Recently, there has been a tremendous boost of interest in personalized services, such as recommendation systems, personal shoppers or targeted advertising. This, in turn, caused a considerable increase in the volume of time-series data, since such systems heavily rely on a continuous collection of user and environment data.

Time-series analysis is a fundamental tool in many domains, such as signal detection, anomaly detection, classification, query, clustering, and forecasting. From the aforementioned, forecasting is, perhaps, the one that received the most attention in recent years, since companies acknowledge that forecasting can be of tremendous aid in the process of decision making. For example, Uber uses time-series forecasting to "predict user supply and demand in a spatio-temporal fine granular fashion to direct driver-partners to high demand areas before they arise, thereby increasing their trip count and earnings" [5]. In our work, we will focus on forecasting and we will study how time-series clustering can improve the predictions. More specifically, we will exploit the similarities between time-series to build better models.

2.2 Time-series clustering

Time-series clustering is an important research topic and many approaches have been suggested for the same in the last decades. The majority of methods used to solve this problem rely on similarity metrics, which are applied either on raw data or on a representation of the time-series. Based on this criterion, the common methods fall into three categories: feature-based approaches, model-based approaches, and raw-data-based approaches [6]. The feature-based methods involve extracting features from the time-series and finding similarities between the features. However, Liao [6] points out that the extracted features are application dependent in most cases, as different sets of features might need to be extracted depending on the use-case. On the other hand, the model-based approaches involve finding a mathematical model that approximates the time-series, based on the assumption that the series have been generated as a mixture of models. Although this method does not suffer from the same drawback as the aforementioned one, finding the best mathematical model can be a very heavy task, which is not desirable when working on data streams. Lastly, as the name suggests, the raw-data-based approaches find groups of similar time-series directly from raw data. These approaches will be the object of our study.

The application which will be presented in this work operates on streaming time-series data. Thus, in the following paragraphs, we discuss only the subsequence clustering, meaning that we will apply the clustering task on non-overlapping windows of data. The choice of non-overlapping windows comes from the literature. Lin, Keogh, and Truppel [7] and Keogh and Lin [8] showed in their studies that clustering sliding windows of time-series data produces meaningless results. The same approach is chosen by multiple studies, such as [9], and will also be adopted by our work.

As for the window size, Mörchen [10] emphasizes the importance of the chosen window size and mentions that the size should be picked according to the application. Our application aims to group users who present similar eating behaviours. Thus, it is crucial to select a window size such that the enclosed information is enough to successfully describe the diet of an individual. Medicine [11] attests that dietary records and diaries are successful dietary assessment tools. The authors suggest that the minimum period required to obtain relevant results is 7 days, and that, usually, the accuracy obtained is directly proportional to the number of days used for the study. Ortega, Perez-Rodrigo, and López-Sobaler [12] suggest that, as a general rule, bigger periods should be recorded for small data sets, while smaller ones are appropriate for data sets with many subjects.

We dedicate the following section to the standard k-means clustering algorithm because our proposed implementation depends on it. Then, we describe the streaming variant that was chosen for this work in. Later, we explain how this standard algorithm fits into the proposed time-series clustering mechanism.

2.2.1 The standard k-means algorithm

K-means is the most popular clustering algorithm nowadays, due to its simplicity and intuitiveness. Firstly proposed by in 1957 and published only in 1982 [13], the k-means algorithm has a simple goal: find *k* centroids corresponding to *k* clusters such that the square distance between each point and its closest centre is minimized. Finding the optimal solution for this problem is known to be NP-hard even for two clusters [14]. However, Lloyd's proposed local search solution, which is described in Algorithm 1, achieves satisfactory results in most cases.

Algorithm 1: The standard k-means algorithm

```
Input: Dataset $D = \{d_1, d_2, \dots, d_n\}$, number of centroids $k$
Output: Centroids $C = \{c_1, c_2, \dots, c_k\}$
1 initialize $C$ with $k$ random points from $D$;
2 do
3     assign each point $d_i$ to the closest centroid in $C$;
4     update each centroid in $C$ by averaging the points assigned to it
5 while centroids change;
```

A high number of research studies have been dedicated to study and overcome the drawbacks of the standard k-means algorithm and to extend it to cover a wide variety of use-cases. We will focus on the following two aspects, as they are the most relevant ones to our study: the scalability of the algorithm when it comes to big datasets and its adaptation for streaming data.

The scalability issues of k-means have their roots in its high execution time when it comes to big data sets. Popular solutions are to parallelize the algorithm by distributing the computation to multiple workers and provide a good approximation of the solution. Dhillon and Modha [15] tackle the problem by proposing a parallel k-means algorithm based on the Single Program Multiple Data (SPMD) model, using message-passing. Zhao, Ma, and He [16] introduce a parallel implementation by leveraging the MapReduce model. The performance of the MapReduce k-means implementation is also evaluated by Golghate and Shende [17], where they highlight the superiority of using a Bulk Synchronous Parallelism computing technique. More recent approaches, such as the one presented by Soliman et al. [18], suggest a decentralized, peer-to-peer k-means approximation.

When it comes to streams, the requirements which should be satisfied by the clustering methods have to change to accommodate the environment. The data cannot be randomly accessed and more strict time constraints have to be imposed. In this case, the most popular approach of adapting clustering algorithms to the data is, similarly to distributed approaches, to provide an estimation of the centroids with approximation guarantees. The clustering process is also changed, as it usually needs two phases: (1) an online phase where the incoming data is received and processed, and (2) an offline phase where the actual clustering process is performed. Moreover, a decision must be made regarding the occurrence of changes in the input distribution over time. Depending on the application, it is debatable whether the algorithm should accommodate new points which, at a first glance, resemble outliers and how much weight should be given to the previously seen points in relation to the new ones.

Another issue which might arise is deciding which data stream points to choose in the online phase when computing the approximation. Some approaches consider all the points which arrive from the stream and use them as mini-batches for the computation [19, 20], while others employ sampling or summarizing techniques [21, 22]. In the following section, we will provide a detailed description of the chosen streaming k-means approach.

2.2.1.1 Streaming k-means with forgetfulness

The method which will be integrated in the pipeline presented in this work is the mini-batch k-means algorithm with forgetfulness [23]. Also used by Apache Spark Streaming [24], a very popular stream processing system, the mini-batch k-means method splits the dataset into buckets and runs the classic k-means algorithm on each of these buckets. Then, a method to allow the model to adapt the centroids over time is applied. This key element is named **forgetfulness**, which allows records from older batches to have an impact over the current centroid computation. The notion of forgetfulness comes by introducing a new parameter, called decay, which adjusts how much weight is given to the previous batches. In each mini-batch, the centroids are computed using the following formulas [25]:

$$C_{t+1} = \frac{C_t \cdot N_t \cdot decay + c_t \cdot n_t}{N_t + n_t} \tag{2.1}$$

$$N_{t+1} = N_t \cdot decay + n_t \tag{2.2}$$

where C_t is the newly computed global centroid at step t (at minibatch t), N_t is the number of total points assigned to C_t, c_t is the local centroid estimated using the points of batch t, n_t is the number of points assigned to c_t from the batch t, and $decay$ is the forgetfulness parameter.

This mini-batch approach is the most suitable for our implementation and requirements because the mini-batching technique matches the windowing approach which will be used by this work. More details about the implementation and design choices will be presented in Chapter 3.

2.2.2 Pattern matching

Broadly, pattern matching is the process of discovering the same subsequence of symbols in two or more different sequences. When it comes to time-series clustering, the patter matching process can be used to find groups of series that exhibit similar patterns. In the case of symbols sequences, the simplest method is to compare each symbol of the series at a given time in pairs. Specific distance measures that are commonly used are the Hamming distance and the Levenshtein distance. For completeness, we define both distance metrics and then motivate our choice.

Hamming distance. Given two one-dimensional arrays $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_n]$, the Hamming distance between x and y is the number of substitutions needed to change x into y:

$$d(x, y) = |\{x_i \neq y_i \mid i \in \{1, 2, ..., n\}\}| \tag{2.3}$$

Levenshtein distance. Given two one-dimensional arrays $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_m]$, the Levenshtein distance between x and y is the minimum number of insertions, deletions, and substitutions needed to transform x into y.

In our study, we cluster together time-series to capture similar trends and seasonality in individuals' diets. We chose the Hamming distance to achieve this purpose. Let us take an example in which the meals are represented by symbols. Given two users u_1 and u_2, let's assume that time t is a Friday and that u_1 ate a meal with representation equal to p, while u_2 ate a meal represented by q. p and q represent very different categories of meals. Let's now assume that, at time $t+1$, on Saturday, u_2 ate a meal which is represented by p. Using Levenshtein distance would yield a higher similarity between the two users than the Hamming distance. However, the diets of these individuals can be very different. For instance, if we consider the meals represented by p as not healthy meals, it might be the case that u_1 has, generally, an unhealthy diet, while u_2 eats such food only during weekends.

2.3 Machine Learning and Deep Learning

Machine Learning (ML) is a subset of Artificial Intelligence in which a system learns how to perform a specific task without explicit instructions, based only on the discovery of patterns. The system has to be capable of analysing a set of data, building a mathematical model, and making decisions when presented with new, unseen data.

Although the term "Machine Learning" exists since 1959 [26] and has been studied ever since, the field has only gained massive popularity recently, as a result of the recent immense growth in data collection. This is a rich mine of knowledge, which, however, cannot be exploited by humans due to two factors: the number of collected entries and their high-dimensional nature. The first problem could be solved in theory by employing a large human workforce who could analyse the data in parallel, but in practice this is not feasible. The second issue is unquestionably unsolvable since human minds are limited to visualizing problems in a maximum of three dimensions. Problems which require a higher number of features become a real challenge.

2.3.1 Core concepts

Machine Learning algorithms are used to solve a variety of problems, from predicting housing prices to detecting skin cancer. Nonetheless, the recipe followed to achieve the goal can always be summarized with the same succession of steps. A high-level view of the pipeline can be observed in Figure 2.1.

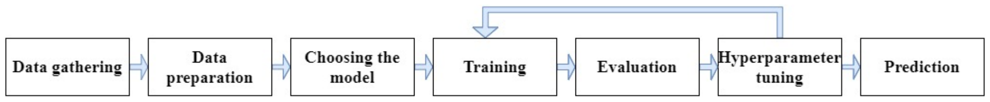


Figure 2.1: Overview of a Machine Learning pipeline

Firstly, the data gathering step ensures the quantity and suitability of the collected data for the given problem. Next, the data is prepared so that it can be used by the ML algorithm. Usually, a small subset of the data, named *validation set*, is kept aside for the evaluation step, while the rest will be used during training (*training set*). Afterwards, as different models are appropriate for different specific tasks, a *model* has to be carefully chosen. The model will, then, *be trained on our data*, meaning that it will learn by iterating over the samples. Usually, the training process is incremental, as the model is sequentially updated and tested to fit the data. Once the training step is finished, an *evaluation step* begins, in which the model is tested against the validation set. The purpose of this step is to evaluate how the model will perform on unseen data. The results of the evaluation determine the performance of the model and can be used to decide if further improvements are needed. If so, a step of hyperparameter tuning begins, in which input parameters of the model are changed and the model is retrained for better results. Lastly, the prediction phase is where the model is applied to the *test dataset*, which is composed of unseen data.

The models are built using *neural networks*. These networks were designed to mimic the behaviour of the human brain and the learning process of humans. The core unit of a neural network was devised to loosely imitate a neuron. This unit is called *perceptron* and is essentially a mathematical function that outputs a weighted sum of the inputs (Figure 2.2).

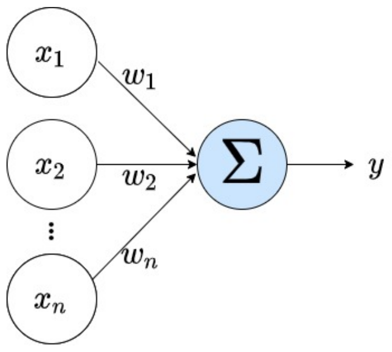


Figure 2.2: The perceptron model. The output y is computed as a weighted sum of the inputs $x_1, x_2, ..., x_n$: $y = \sum_{i=1}^{n} w_i \cdot x_i$

The goal of a neural network is, usually, to approximate a function f. The training process is designed to find a function $\hat{f}$ that is as similar to f as possible. Thus, it uses the data in the training set and iteratively tweaks the weights of the network until $\hat{f}$ is as similar as possible to f. This similarity measure between the two functions is commonly measured using a *loss function*, which quantifies the difference between f and $\hat{f}$. The choice of a loss function depends on the problem.

The perceptron, albeit simplistic, can be used to solve very complex problems, if used appropriately. This goal is achieved by *deep neural networks*, which are composed of multiple layers of perceptrons stacked together. Although the training algorithms and technologies, deep learning has seen massive success only recently due to two factors: the amount of data available for training increased drastically and the progress of graphics processing units, which now can be exploited to train deep networks in a reasonable amount of time, even for huge datasets. The classic example of deep neural network that achieved exceptional results is GoogLeNet [27], which was trained to perform image classification. This modern architecture stacked more than 30 layers on top of each other to achieve a classification error of only 0.06656. Similarly, deep architectures proved to be highly efficient in a multitude of fields, such as speech recognition, natural language processing, medical image analysis and so much more.

The first designed deep neural network, called Feedforward Neural Network (FNN), is built simply by stacking multiple perceptron layers. In this type of architecture, the information travels in only one direction, from inputs to output. In the following subsection, we will briefly present more sophisticated architectures that are commonly used for time-series forecasting.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a special variety of artificial neural networks that are suitable for sequential data. Feedforward neural networks output an answer based on the current input. However, RNNs aim to provide an answer which takes into consideration not only the input, but also the previous states of the network. The previous state of the network is referred to as memory and is the key element in the architecture of a RNN. This unique trait has been proven to be very successful in a wide variety of applications, such as language translation, handwriting recognition or speech recognition.

The name of Recurrent Neural Networks comes from the nature of their architecture. As the output is constructed based on both the input and the previous state, the update rule is a recurrence relation:

$$h^t = f(h^{(t-1)}; x^t; \theta) \tag{2.4}$$

where h^t is the hidden state at step t which is a function f of the previous hidden state $h^{(t-1)}$ and the current input x^t, and θ represents the parameters of the function. As can be noted, the internal state at step t depends on the internal state at step $t-1$, which in turn depends on the internal state at step $t-2$ and so on. These loops allow the information to persist inside the network, meaning that the output will use historic data for each prediction. The unfolding of this equation can be noted in Figure 2.3.

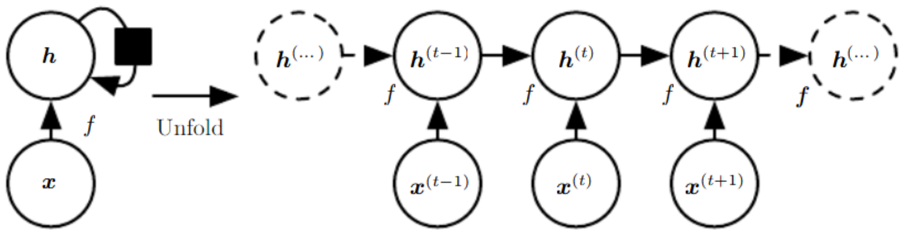


Figure 2.3: The unfolding of a Recurrent Neural Network with no outputs. Source: Goodfellow, Bengio, and Courville [28], page 370

RNNs can be classified into four categories, based on the shape of their input, output, and the problem they aim to solve. The first type, sequence-to-vector, takes a sequence as input and overlooks all the outputs, except the last one. The sequence-to-vector RNN can be used, for example, in assigning sentiment scores for sequences of words. Similarly, the vector-to-sequence RNN takes a single input and produces a sequence. This type of architecture can be used in applications such as image captioning. Sequence-to-sequence networks take sequences as input and produce sequences as output. These networks are especially useful in predicting time-series. Lastly, the encoder-decoder RNNs are composed out of two networks: a sequence-to-vector network which acts as an encoder, and a vector-to-sequence network which acts as a decoder. The encoder-decoder architectures are used in problems such as inter-language translations. As our goal is to do time-series predictions, we will only focus on the sequence-to-sequence architecture.

Although the architecture of the RNNs is successful in a variety of use-cases, these networks are susceptible to problems when it comes to long-term dependencies. It has been proven that RNNs are very effective when the prediction needs only information from the recent past. However, when the needed information for a prediction is at a considerable distance in the past, RNNs have been proven to have difficulties in making connections. The reason behind this problem comes from the recurrence of the update rule itself. Over time, the gradient of the loss functions decays drastically, causing the long-term dependencies to be lost [29]. The Long-Short Term Memory networks are modified RNNs that solve the vanishing gradient issue.

2.3.2.1 Long-Short Term Memory

Introduced by Hochreiter and Schmidhuber [30] in 1997, the Long-Short Term Memory networks (LSTMs) are, essentially, Recurrent Neural Networks redesigned to overcome the vanishing gradients problem. Nowadays, the popularity of LSTMs increased tremendously and are usually one of the default choices when it comes to classifying and predicting sequences of data [31].

The main idea of LSTMs is making the network decide which information is relevant and which information can be forgotten. The only difference between LSTMs and RNNs is the computation of the hidden state of the network. While in RNNs hidden state is represented by only one vector, in LSTMs the hidden state is split into two vectors $h^{(t)}$ and $c^{(t)}$, which act as a short-term state and a long-term state, respectively. Three special gates control which information is kept and which one is erased at each step: the **forget gate** which decides which information should be thrown away, the **input gate** determines what new input information x^t is useful and should be added to the state, and the **output gate** that builds the output. A simplified view of the LSTM cell can be observed in Figure 2.4.

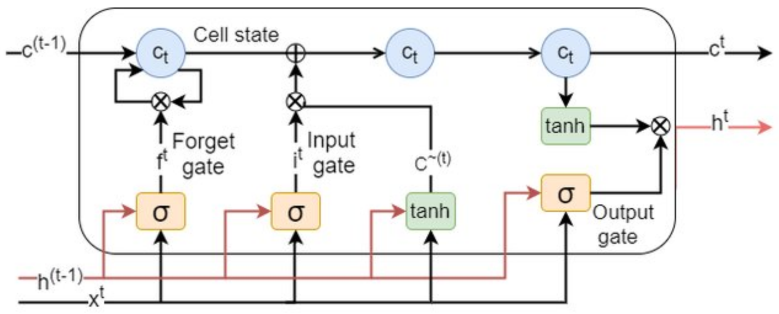


Figure 2.4: Long Short Term Memory cell. The symbol σ is used to denote the sigmoid function, while the symbols $\otimes$ and $\oplus$ represent element-wise operations. Source: Jenkins et al. [32]

The architecture of LSTMs can be simplified by combining the forget gate and the input gate into only one operation. This modification of LSTMs is a popular variant called Gated Recurrent Unit (GRU). GRUs were introduced by Cho et al. [33] in 2014 and started gaining popularity recently. GRUs result in a simpler model than the LSTM models and, with similar results in a number of fields, such as music and speech modeling. Moreover, it has been shown by Chung et al. [34] that GRUs might outperform LSTMs for certain small datasets. However, Britz et al. [35] performed an extensive analysis of Neural Machine Translation Architectures and proved that LSTM cells are superior to GRU cells in most cases.

In Chapter 5 we perform a search for the model with the highest performance for our use-case and do a brief comparison between LSTMs and GRUs on our dataset.

2.4 Federated learning

Federated learning is a novel approach on distributed machine learning, which takes advantage of the recent advances in the capabilities of modern mobile devices. The two most appealing characteristics of federated learning are the gains in privacy and performance [36, 37].

Recently, machine learning areas such as speech recognition, computer vision, and natural language processing have seen major breakthroughs. One of the main reasons that caused this success is the explosion in the quantity of data that is available for training. Companies have now started to acknowledge the necessity of data collection more than ever, and gigantic datasets have been made available. However, learning from such voluminous datasets can be a costly task. Moreover, collecting and learning from raw data can be problematic when the data contains privacy-sensitive information. Federated learning aims to solve both of these issues by building a federated model without having access to local data.

Let us consider a real-life scenario. Several clients are using their mobile devices, on which individual user data is stored. In federated learning, we learn from the data of all the participants without ever seeing the actual raw data. The actors of the federated learning algorithm are the *clients* and a *central coordinator* (often called *server*). Each client holds a local dataset which contains only their data. This dataset will never be shared either with the server or with other clients. The whole process is described using Figure 2.5. The server shares a central model with all the clients. Then, each client improves the current model using information from their local data (A) and sends the update back to the server. The server aggregates the updates from multiple users (B) and an improved central model is created and shared with the clients (C). The process repeats as clients' mobile devices collect more data.

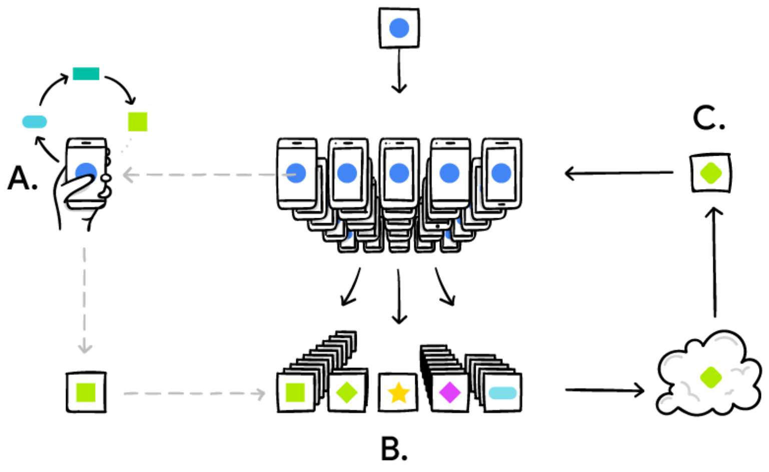


Figure 2.5: The federated learning algorithm. Source: McMahan and Ramage [38].

One of the first examples of federated learning was solving the task of predictive keyboards. Hard et al. [39] show how federated learning has been integrated to train a Recurrent Neural Network to perform next-word prediction for Google keyboards. Another use-case comes from the medical field, in which Brisimi et al. [40] predicted future hospitalizations for cardiac events. Samarakoon et al. [41] applied federated learning to improve vehicle-to-vehicle communication, while Zhao et al. [42] designed a system to predict the requirements and consumption behaviours of IoT devices customers using federated learning.

Unlike traditional distributed learning approaches, the federated optimization has several key characteristics that are very appealing to use cases such as the ones mentioned in the previous paragraph. These properties are essential to our study since our data and environment are the perfect matches for the federated learning algorithm:

- **Non independent and identically distributed (Non-IID)**: As the training data is local on each user's device, it is expected that any local dataset will not be representative of the whole population.
- **Unbalanced**: Each person uses their device differently. Hence, it is expectable that there will be a big variance in the size of local datasets.

- **Massively distributed**: The number of participants should be much larger than the number of observations per participant.
- **Limited communication**: The communication might not be possible with all devices at the same time.

However, federated learning comes with a series of known drawbacks. This technique assumes a minimum computational power for each device and requires good connection between the server and the participants, which might not always be achievable. Moreover, the results obtained by training a global federated model might be suboptimal if the distribution of clients' data diverges. Lastly, even if federated learning offers privacy gains by design, the exchange of parameters between the participants and the server might lead to leakages of sensitive data. In the related work section, we will present solutions found in the literature for the last two problems, as they will be the most relevant to our study.

2.5 Differential privacy

Nowadays big companies shifted their focus towards personalized services, which allow tailoring the user experience to match each individual's preferences. To achieve this goal, companies started collecting more and more user data to improve their services. However, some information can be sensitive and can lead to harmful outcomes if leaked. Thus, a trade-off must be found between companies needing large amounts of data and the users who demand data privacy.

The most intuitive way of hiding user data is data anonymization, a process in which personal data is either encrypted or removed completely from the dataset. This method has been proved to be unsuccessful. Perhaps the most popular example is the case of the Netflix prize dataset. In late 2000s Netflix started a competition to find the best algorithm to predict user ratings for movies. They published a dataset which contains movie ratings that were anonymized by removing the usernames completely and replacing some of the ratings with random values. Unfortunately, the anonymization turned out to be ineffective, as Narayanan and Shmatikov [43] successfully identified the users in the dataset by combining the Netflix data with information available on IMDb.

Differential privacy (DP) is a technique that allows companies to aggregate and query user data without compromising the privacy of the involved individuals [44]. The algorithm comes with a powerful guarantee. Let's assume a user U has doubts about whether they should allow their data to be collected or not. Differential privacy ensures that any query on a dataset D containing U's information will have the same result as running the query on a copy of D which does not contain U's data. In other words, adding U's information to the dataset should not affect the outcome of the query, meaning that the query cannot be used to expose U's sensitive information. The key to achieving this property is the addition of *noise*. The most popular noising methods of achieving DP are sampling the noise from either a Gaussian or Laplacian distribution.

Differential privacy comes in many flavours. However, the most popular mathematical formula used to express it is as follows [45]: given a randomized algorithm A, the set of all datasets D and D' that differ on at most one row (i.e. the data of one individual), and any subset $S \subseteq range(A)$,

$$Pr[A(D) \in S] \leq e^{\epsilon} Pr[A(D') \in S] \tag{2.5}$$

The epsilon (ϵ) parameter is used to quantify the loss of privacy. As can be noted from the formula, absolute privacy is obtained when ϵ equals 0. Another aspect that should be mentioned is that achieving higher levels of privacy involves adding more noise to the data which leads to a decrease in the accuracy of the algorithm. A trade-off should be found between keeping the information as private as possible and achieving meaningful and accurate results.

Two known limitations of differential privacy should be mentioned. Firstly, DP can usually be applied with success only on large datasets, as adding noise to very small datasets will likely result in inaccurate results. Secondly, DP is vulnerable when it comes to repeated queries. A malicious adversary can gather data that is close, but not equal to, the ground truth. However, after multiple attempts, the adversary might be able to estimate the ground truth. To this end, the privacy level has to be tweaked to ensure tolerated information loss and increased privacy.

2.6 Apache Flink

Apache Flink [46] is an open-source framework devoted to processing batch and streaming data. One of main characteristics of Flink is that it processes batches as a particular case of streams, allowing streams to be treated as continuous, time-dependent flows of data. This characteristic makes Flink a true stream processing system.

Apache Flink is a distributed processing system which supports stateful computations over both batch and streaming data. It provides fault-tolerance by integrating a light-weight mechanism based on checkpoints and recovery. This mechanism ensures exactly-once state consistency. Moreover, Flink provides support for highly scalable applications, as it has been designed to distribute the state to an arbitrary number of available workers and to run in all common cluster environments.

The most popular use-cases of Apache Flink are event driven applications, data analytic applications, and data pipeline applications [47]. The system which will be introduced by this work falls into the latter category.

In the following subsections, we will focus on the DataStream API, which is used for processing unbounded streams of data, and on iterative processes over data streams in Apache Flink.

2.6.1 The DataStream API

Apache Flink is designed to process both streaming and non-streaming data. For this, it provides two APIs, the DataStream API and the DataSet API. As Apache Flink treats batches as a particular case of streams and our work is geared towards streams, we will only outline the DataStream API.

In a Flink program, the input data is read from a data source. Flink currently supports a multitude of streaming data sources, such as file-based sources, TCP sockets, collection-based sources, and custom sources (Kafka, user-defined sources etc.).

Processing a data stream is performed using operators, which convert one or more data streams into another stream, by applying transformations like map, filter, fold or union on the stream records. Apache Flink also supports windowing operators, which are one of the key elements in the ML pipeline that will be presented by this work. The window operators split the stream records into buckets, usually based on temporal boundaries with respect to the event or processing time of the records. Operations can be applied on windows and will be triggered once the window is filled. Flink supports splitting the stream into tumbling, sliding, session or global windows.

2.6.2 Iterative data streaming

Iterations are a key concept in many algorithms, especially in fields such as Machine Learning or Graph Analysis. Apache Flink natively supports iterative processes for the DataSet API, but not for streaming data. Recently, Carbone

[48] added support for arbitrary nested iterations on top of native Flink single-pass stream processing.

The newly proposed looping operator runs iterative processes on windows. The process follows the Bulk Synchronous Processing (BSP) model for distributed computing [49]. More specifically, the computation is structured in supersteps in which distributed processes execute local instructions and modify their local state. In addition, there is a router abstraction that routes messages between any two processes, and a synchronization primitive that blocks all processes' activity until all of them complete the current superstep.

The iterative operator on windows in Flink is described by the following components: (1) a termination criterion, (2) a synchronization option, (3) a key that partitions the input data, and (4) three core aggregation functions (entry, step, and finalize), which are called accordingly during the iterative process. Moreover, the model manages two types of state: a loop state per window that is cleaned at the end of one iterative process, and a persistent state, which contains state accessible across all windows. Figure 2.6 presents a simplified view of the internals of the window iterate operator.

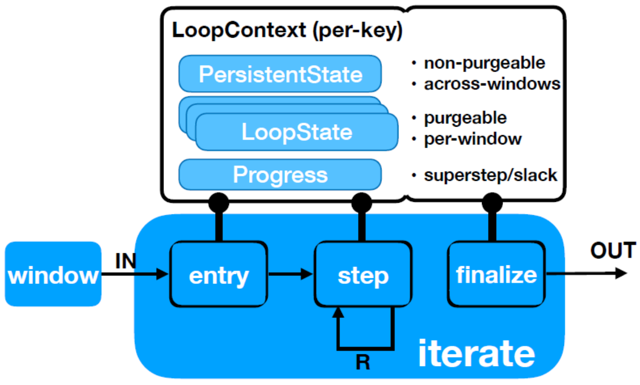


Figure 2.6: The internals of the window iterate operator in Apache Flink. Source: Carbone [48].

Our clustering mechanism is highly dependent on iterations for streaming data. Thus, this model is a key element for the system which will be presented in the following chapters.

2.7 Related work

In this section, we discuss similar approaches found in the literature and compare them to our solution. To the best of our knowledge, there is no other work that aims to investigate the problem of dietary and user health forecasting in a streaming context at the moment of writing this thesis. Thus, in the following subsections, we present works that are comparable to each of the steps chosen in our pipeline.

We first discuss similar research that has been done in the field of multivariate time-series clustering. We present the approach that inspired our mechanism: a two-phase mechanism. As the second phase in our proposed implementation is trivial, we then focus on the first one, in which we aim to discretize the data. Then, we present the streaming k-means variant that has been chosen for the discretization process.

Lastly, we address the two drawbacks of federated learning that will constitute the object of our study: clustered federated learning and differentially private federated learning. We mention work that has been undertaken in literature and compare these approaches to ours.

2.7.1 Multivariate time-series clustering mechanism

Our work is inspired by the work proposed by Liao [50]. Their clustering technique is composed out of two logical steps: a step in which the real-valued data points are converted into discrete univariate time-series, and a second step in which a clustering procedure is applied on the resulted univariate time-series to obtain the final results. The author uses fuzzy c-means for the first step and maps each real-valued data point to a discrete value. The symmetric version of Kullback–Liebler distance is chosen for the second step. The author of this study mentions that the proposed method provides a general framework that can be easily adapted depending on the use-case.

Our work will adopt this framework, by using different algorithms for each step. Specifically, we will implement a streaming variant of the k-means version for the first step, followed by computing the Hamming similarity for the second step. We will refer to the first step as the discretization step, while the second step will be referred to as the pattern matching step. As the second step is trivial, in the following sub-sections, we focus discuss similar approaches found in the literature on the topic of time-series discretization.

2.7.1.1 Discretization of time-series

This work will use a method to discretize time-series which is very similar to the one proposed by Das et al. [51]. The goal is to map each multivariate data-point of a time-series into a discrete value. The authors introduced one of the most well-known and cited sub-series clustering algorithm, in which they suggested that the shape of the pattern is the most important trait of a series, while the actual values can be neglected. In their study, the real-valued time-series was split into windows. Then, the series is discretized and the problem translates into the discovery of rules in symbolic series. Each subsequence is clustered using k-means. The discretized version of the time-series is obtained by using the identifier of the cluster with the closest centroid to each subsequence. Lastly, the series of symbols are clustered using a suitable similarity measure.

It has been proven that this method can be unsuccessful in some cases. A degree of information is lost in the discretization process, as the centroids obtained by k-means can fail to effectively represent the data. Thus, some studies suggest that this method can lead to mining of meaningless rules [7, 52]. However, the state-of-the-art work proposed by Shokoohi-Yekta et al. [52], which operates directly on raw real-valued time-series, may also result in invalid rules and is not generally applicable to multiple time-series. Moreover, the symbolization method proposed by Das et al. [51] is attractive for our case because of two main reasons: (1) working with low-dimensional symbols decreases the execution time of the algorithm, which is crucial in low-latency systems such as streaming applications, and (2) working with symbols instead of raw data brings a benefit in terms of privacy.

Our work is also inspired by the work proposed by Chen et al. [53], in which the authors aimed to discover similar patterns in the travelling habits of the subjects over streaming trajectories. Their approach partitions the stream into windows, applies a clustering algorithm to observe the movements of the individuals in the given time frame, and uses the clusters to detect co-movement patterns. The main idea is to detect the individuals who have similar travelling itineraries, meaning that individuals who often follow similar trajectories should be detected as belonging to the same group. However, locations are not well-defined points in space and discerning whether two people visited the same spot can be tricky. For that, the authors cluster the locations with respect to a specific threshold (Figure 2.7). Since this paper uses geographical information from GPS records, we cannot apply the same mechanism on our use-case. However, we follow the same principle, with a few tweaks that suit our problem.

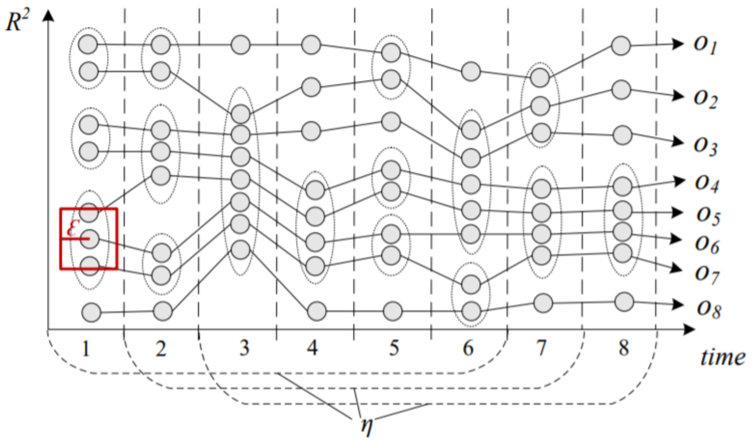


Figure 2.7: Example of co-movement pattern. The figure depicts the patterns of users $o_1, o_2, ..., o_8$, given a window of size η and a clustering algorithm that groups together locations in a range equal to ϵ. Source: Chen et al. [53].

2.7.2 Federated learning approaches

This section focuses on studies that aimed to solve two of the drawbacks of federated learning: the shortcomings of training only one federated model that should provide accurate predictions for all the participants, and the need for additional privacy methods so that sensitive data cannot be inferred from the updates sent to the server. Thus, this section is organized around these two problems. We first discuss common approaches in the literature for the former, which are based on the idea of training separate models for different groups of users. Then, we present approaches that solve the latter by introducing differential privacy techniques to the federated learning algorithm.

2.7.2.1 Clustered federated learning

Sattler, Müller, and Samek [54] introduced in their work the Clustered Federated Learning (CFL) framework to improve the performance of classic federated algorithms by leveraging natural groups that exist in the client population. The concept has been adopted by several studies, such as Huang and Liu [55], who applied a clustering technique on patients' data to boost the performance in predicting the hospitalization time and mortality using electronic medical records. More, Dıaz González [56] studied time-series forecasting and showed that training separate global models for different clusters of time-series improves the local performance.

The procedure is not very different from classic federated learning. First, the server clusters the clients according to a chosen similarity metric. Let's assume it discovered k groups. Then, the federated learning algorithm proceeds with the small modification that the server maintains k models instead of one. When an update from a client arrives, the server must first check the cluster label of the client to decide to which model the user's update will contribute to. Similarly, each client has to download the model corresponding to its cluster. As this method was also adopted by our work, more details about the algorithm will be presented in Chapter 3.

2.7.2.2 Differentially private federated learning

As described in Chapter 2.4, federated learning made a significant step towards better privacy protection by offering a training mechanism that can learn from clients' data without having access to such data. However, sensitive information can still be inferred from the updates sent to the central server. Zhu, Liu, and Han [57] proved in their study that it is, in fact, possible to obtain such information from shared gradients.

Other aggregation algorithms need only the weights that were obtained after training the local model. Studies such as the one undertaken by Ma et al. [58] or Shokri and Shmatikov [59] show that this technique is not safe either and that private information of individuals can still be divulged.

Regardless of the parameters chosen to be shared, each new communication round in a federated learning algorithm can lead to data leaks, which accumulate in time throughout the process. Differential privacy can, thus, be used to conceal the contribution of each client during the training process. Similarly to the general idea of DP, a trade-off must be found between privacy loss and model performance.

Geyer, Klein, and Nabi [60] address the problem of differentially private federated learning from a client point of view. Their proposed algorithm distorts the updates sent at each communication round by adding Gaussian noise. The experiments suggest that this method can maintain the privacy of the clients with only a small decrease in performance, given the dataset size is large enough. Similarly, Choudhury et al. [61] tackle the same problem in the context of sensitive health data. In their work, the authors add noise to the optimization function and prove this approach offers good data privacy while maintaining an adequate model performance. In our work, we will also apply this DP technique and study its effect on the accuracy of the model.

Lastly, we will also study the effect of another differential privacy method, which involves disturbing the training datasets themselves by adding noise. The study that served as our main inspiration for this DP algorithm has been undertaken by Li et al. [62], in which the authors used DP for stock price predictions. Similarly to this work, they tackle the problem of time-series forecasting, as they aimed to predict the stock prices. Albeit in a non-federated context, this study is closely related to our work because it noises the data directly to achieve better privacy preservation.

Chapter 3

Design and implementation

The main goal of this chapter is to present the proposed pipeline, describe each different sub-system, and how they interact with each other. The main components of the pipeline are: (1) the clustering mechanism, which includes the streaming k-means sub-system and the pattern matching sub-system, and (2) the system of federated learning with privacy guarantees.

Our proposed pipeline consumes a stream of time-series diet and health logs from several users and attempts to predict the next logs by leveraging the history of each individual and the similarities between users. Given the increasing interest in health self-monitoring, the data load can be considerable. Thus, the system has to be highly scalable. Besides, the streaming nature of the problem imposes strict requirements in terms of latency, as the system should be able to provide real-time predictions. Last but not least the proposed pipeline has to ensure a strict level of privacy preservation because we are dealing with privacy-sensitive data.

Firstly, we cluster the users and use this information to build separate federated models for each group. In this way, we ensure that the prediction caters for individuals with unique dietary patterns and lifestyles, thus offering personalised forecasts. We introduce our two-step clustering mechanism, that is able to group multidimensional time-series data based on common characteristics.

Secondly, we address the problem of distributed time-series forecasting in a federated learning context. We pick the best model architecture for our use-case and we motivate the choice of federated learning for our scenario. Then, we discuss the implementation details and decisions of applying further data protection methods to the federated learning algorithm.

3.1 Clustering dietary habits and user health

The first essential component of our pipeline is the clustering task. Given a stream of user logs, we aim to identify the groups of users with the most similar habits.

We cluster the users using a **two-phase mechanism**. For the first phase, we adapt the mini-batch k-means with forgetfulness algorithm presented in Chapter 2.2.1.1 to match our streaming context. It is important to note that, for this step, we omit any temporal information, as we aim to discover the most representative centroids in the dataset. In the second step, we extract the patterns of individuals using the information from the first phase and the timestamps of the logs. Then, we attempt to group the ones with the most similar patterns. The first step can be configured as a monitoring technique, that will give insights into the overall shape of the data, while the second can be triggered only at times when reconfiguration is necessary.

3.1.1 Streaming mini-batch k-means

For the first phase of our clustering method, we modify the k-means algorithm with forgetfulness so that it can accommodate our scenario. We design a clustering pipeline on streaming data and implement it using the novel Iterative Data Streaming framework of Apache Flink [48].

In general terms, the objective of this system is to apply the k-means algorithm on streaming data and obtain relevant results, while distributing the data load among several workers. The proposed implementation is not limited to our specific use-case and can be applied to all clustering problems that are suitable for k-means.

The main idea of the implementation is based on the Iterative Data Streaming framework. First, we split the stream into non-overlapping buckets of fixed and finite size, named tumbling windows. Then, we run k-means on each of these consecutive windows and use the forgetfulness parameter to account for changes in data over time. The mini-batch k-means algorithm matches our case perfectly, as the tumbling windows act as mini-batches.

The scalability requirement of our implementation is achieved by distributing the load among a number of workers. Each worker applies the algorithm on a disjoint subset of the data (accumulated in the given time window) and communicates with the other workers by exchanging messages until they reach consensus. In our case, the message passing stops when all the workers agree on a set of centroids, which will constitute the output of this sub-system.

Figure 3.1 depicts a high-level view of the proposed streaming k-means pipeline. The system is split into three steps, which will be explained in detail in the following paragraphs.

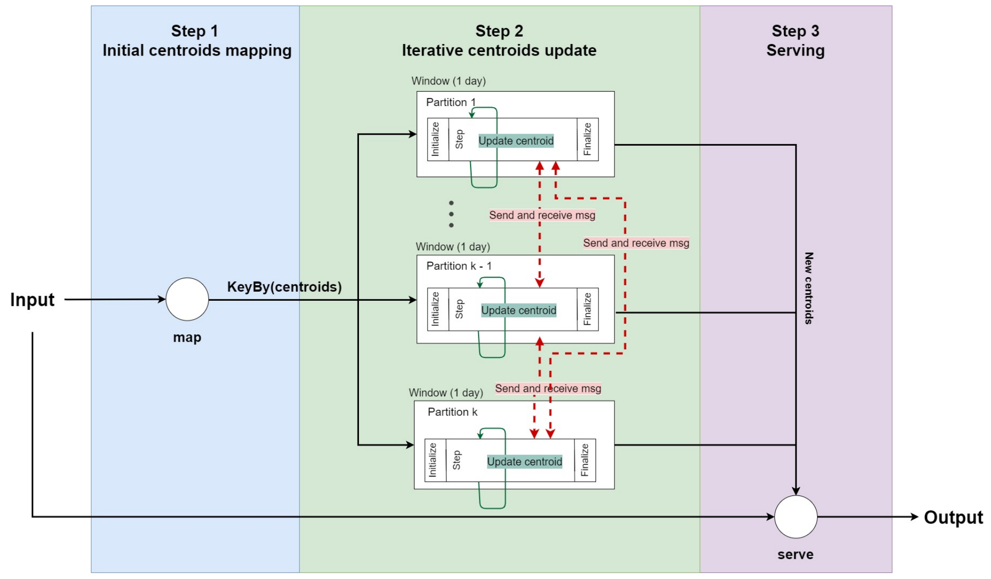


Figure 3.1: The proposed streaming k-means pipeline

3.1.1.1 Input, output, and parameters

As can be noted in Table 3.1, the algorithm takes as input a stream of records which contain a unique identifier *id* and the features contained in the log ($[< features >]$), such as the nutritional breakdown, resting heart rate, and active minutes. It produces a stream in which each of the input record is assigned to a specific centroid *c*. Naturally, the number of clusters *K* has to be pre-defined, along with the forgetfulness parameter *decay* and the size of the window $size_w$.

Table 3.1: Input, output, and parameters of the streaming k-means implementation

Input	stream of $(id, [< features >])$ records
Output	stream of (id, c) records
Parameters	K
	$decay$
	$size_w$

3.1.1.2 Step 1 - Initial centroids mapping

The first step of the k-means pipeline has two goals: to initialize the centroids of the clusters and to partition the stream into disjoint sub-streams.

The internal logic of this step is depicted by Algorithm 2. Intuitively, the first K unique records that come from the stream are chosen as initial centroids (line 1). Then, each of the records that arrive is matched with one of the initial centroids. This is achieved by using the $findClosestCentroid()$ method (line 3), which takes a stream record and a list of centroids and returns the closest centroid from the list with respect to the given record. Based on this matching, the stream is keyed by the initial centroids, resulting in K disjoint partitions (line 6).

Algorithm 2: The k-means initialization algorithm

```
Input: Stream of $r_i = (id_i, [< features >])$ tuples
Output: Stream of $(c, r_i)$ tuples, keyed by $c$
1 $I = \{r_0, r_1, \ldots, r_{K-1}\}$ the first $K$ stream records;
2 foreach $r_j$ with $j \geq K$ that comes from the stream do
3     $r_p = findClosestCentroid(r_j, I)$;
4     output $(p, r_j)$
5 end
6 key the output by the centroid id;
```

One essential aspect which should be noted is that each centroid carries a **unique identifier** which is assigned on initialization. This identifier is used to key the stream and will be an important feature for the following sub-systems. In addition, it is important to note here that this approach can benefit from a maximum parallelism of *K*, since that is the number of resulted disjoint partitions.

3.1.1.3 Step 2 - Iterative centroids update

The core of the clustering sub-system lies in the task of iteratively updating the centroids. In this step, the objective is to apply k-means on the data captured in each time window.

The second step takes as input the output of the previous step. Thus, we have *K* partitions of the original stream and each partition is responsible for one centroid, which means that each worker updates its centroid. For clarity, in the following paragraphs we will refer to the partition responsible for the *i*th centroid as the "*i*th partition" or "*i*th worker". The partitioned streams are windowed into batches of equal size. For each of this batch and each of these partitions, we apply a modified version of the mini-batch k-means algorithm with forgetfulness.

The Iterative Data Streaming model in Apache Flink [48] builds iterative processes over windows by following the Bulk Synchronous Processing computing paradigm. Our update of centroids complies with this model. Firstly, we partition the input stream and key it by the closest initial centroid. Secondly, we apply the iterative process over keyed time windows and organise the iterations as parallel steps (**supersteps**), in which each new iteration is allowed to start only after the previous iteration was completed across all partitions. The partitions communicate during the iterations by message passing. Lastly, the iterative process is split into three core aggregation functions: **entry**, **step**, and **finalize**.

Algorithm 3 describes the flow of the iterations. We start by motivating how we adapted the mini-batch k-means algorithm to achieve our goal. Then, we define the messages exchanged between the workers and the methods used as building blocks for the iterative process. Finally, we go through the whole algorithm to explain the idea behind it.

It is important to understand how the classic mini-batch k-means algorithm [23] is mapped onto our setting. In the original algorithm, the mini-batches are found by sampling from the original data set. Then, a set of centroids is computed for each mini-batch. At the end of the computation of these local centroids, they are merged into the global centroids using the forgetfulness parameter. To adapt this scenario to our case, we think about the windows as random samples of the data set and we use the concepts of **loop state** and **persistent state** (which exist in the Iterative Data Streaming model) to account for the local and global centroids. Thus, in the loop state (line 2) we compute and store the local centroids, which will be purged after each window, while in the persistent state (line 3) we compute and store the global centroids.

Our algorithm reaches consensus using the message passing paradigm. We define two types of messages, *update* and *shuffle*, as it follows:

$$update(i, c) \text{ with } i \in \{0, 1, ..., K\} \text{ and } c \text{ centroid} \tag{3.1}$$

$$shuffle(i, M_i) \text{ with } i \in \{0, 1, ..., K\} \text{ and } M_i \text{ the list of records} \tag{3.2}$$

The messages of type *update* are exchanged when the *i*th partition updated its centroid. In this case, the partition sends an *update* message to all the other partitions, letting them know about the new value of the *i*th centroid. The messages of type *shuffle* are used to exchange records which no longer belong to certain partitions. A *shuffle(i, M_i)* message is sent to the *i*th partition with all the records that were found to belong to the *i*th centroid (M_i).

Table 3.2 defines the methods used as building blocks. We chose to only briefly describe their purpose and not to dive into explicit details about their implementation because these methods are rather simplistic and self-explanatory.

Table 3.2: The methods used as building blocks for the second step of the streaming mini-batch k-means pipeline

Method	**Description**
$assignClosestCentroid(M, L)$	given a list of records M and a list of centroids L, assign each record in M to the closest centroids in L and output a list $\{M_1, ..., M_{K-1}\}$ where M_i is the list of records from M which were assigned to the ith centroid in L
$computeNewCentroid(M)$	given a list of records M, computes a new centroid by averaging all the values in M
$updatePersistentState(L, P)$	updates a list of global centroids P using a list of local centroids L and the forgetfulness parameter

Last but not least, we go through the algorithm and explain the steps. For each time window and each partition, the process is identical. As mentioned before, each partition is **responsible for one centroid**. The algorithm aims to compute local centroids for each time window and store them in the loop state *L* and use these centroids to update the global centroids stored in the persistent state. A partition *i* uses only the *i*th position in *L* to compute its centroid. At the end of each time window, all partitions will reach consensus and will output the global centroids. The computation is organised into three phases: **entry**, **step**, and **finalize**.

We start with the **entry** phase, in which we initialize the local state. If the time window is the first one, then the local centroid corresponding to the *i*th partition is initialized with the *i*th initial centroid chosen at **Step 1**. Otherwise, the *i*th local centroid is initialized with the *i*th global centroid that was computed in the previous time windows (lines 7-8). At the end of this phase, each partition sends an *update* message to all the other partitions to announce which initial centroid was chosen (line 9).

Consequently, the **step** phase starts by reading all the messages that were received. If the ith partition received a *update(j,l'_j)* message, then it updates the jth centroid in its local state with the new one (line 12). With this step, each worker has updated knowledge about the centroids of the other workers. Let M_i be the records assigned to the ith centroid so far. If the received message was a *shuffle(i, M_i')* message, then the partition received more records which belong to its centroid. Thus, it adds these new records to M_i (line 13). After line 14, the ith partition has up to date information regarding the centroids of the other partitions and the records that belong to its centroid. This is guaranteed by the Bulk Synchronous processing model. Using the information about all the centroids and the local records, the ith partition re-assigns its local records to the closest centroids (line 15). Then, it updates its centroid by averaging the records that still belong to the ith centroid in L (line 16). If the centroid changed, then the ith partition sends an *update* message to announce that its centroid changed (line 18) and sends *shuffle* messages to forward the records that are not in the ith cluster anymore to their corresponding partitions (lines 19-20). If the centroid did not change, no new messages are exchanged. This phase is repeated until no more messages are sent and received, in which case the centroids converged and the partitions reached consensus. After this phase, the loop state L contains the local centroids computed for that time window.

In the last phase, called **finalize**, the loop state L is used to update the persistent state P (the global centroids). Then, the loop state is purged and each partition outputs the global centroids P. Since they reached consensus in the previous phase, all of the partitions updated the global centroids in the same manner, causing them to output the same set of global centroids.

Algorithm 3: The k-means iterative update of centroids

Input: Stream of $(c, (t_i, [< features >]))$ tuples, keyed by c

Output: Stream of $C = \{c_0, c_1, \ldots, c_{K-1}\}$ centroids

```
1  split the input stream into **tumbling windows** of size ***size***$_w$;
2  let $L = \{l_0, l_1, ..., l_{K-1}\}$ be the loop state;
3  let $P = \{p_0, p_1, ..., p_{K-1}\}$ be the persistent state;
4  foreach time window do
       /* Entry */
5      let $i$ be a key of the keyed stream, $r_i \in I$ (from Step 1);
6      let $M_i$ be the set of records which were assigned to $r_i$ at Step 1;
7      if $P \neq null$ then $l_i = p_i$ ;
8      else $l_i = r_i$ ;
9      send $update(i, l_i)$ message;
       /* Step */
10     do
11         foreach received message $m$ do
12             if $m == update(j, l'_j)$ then $l_j = l'_j$ ;
13             else if $m == shuffle(i, M'_i)$ then $M_i = M_i \cup M'_i$ ;
14         end
15         $[M_0, M_1, ..., M_{K-1}] = assignClosestCentroid(M_i, L)$ ;
16         $r'_i = computeNewCentroid(M_i)$ ;
17         if $r_i \neq r'_i$ then
18             send $update(i, r'_i)$ message;
19             foreach $M_q \in [M_0, M_1, ..., M_{K-1}]$ do
20                 send $shuffle(q, M_q)$ message;
21             end
22         end
23     while messages are being exchanged;
       /* Finalize */
24     $P = updatePersistentState(L, P)$;
25     purge $L$;
26     output $P$;
       // apply the same steps for all keys, in parallel
27 end
```

3.1.1.4 Step 3 - Serving

The third and final step of this sub-system is the serving part, which is straight-forward. As noted in Algorithm 4, the serving part takes two streams as input: the original raw input stream which contains user logs, and the output of Step 2, which consists of the updated centroids. By design, the latter is considerably slower than the first, as the frequency of the centroids set updates is determined by the size of the window used at Step 2 and the computation time necessary for the operation.

Thus, at Step 3 we store the latest centroids computed in the previous step. Then, we assign each new record which arrives from the stream to a centroid, by applying the $findClosestCentroid()$ method (line 4), and we output the id of this centroid.

Algorithm 4: The k-means serving algorithm

```
Input: Stream of $r_i = (t_i, [< features >])$ records
       Stream of $C = \{c_0, c_1, \ldots, c_{K-1}\}$ centroids
Output: stream of $(t_i, p)$ tuples, with $c_p \in C$
1 let $C$ latest updated centroids;
2 foreach $r_i$ new record do
3     $c_p = findClosestCentroid(r_i, C)$;
4     output $(t_i, p)$
5 end
```

3.1.2 Finding similar patterns in the data

The aim of the subsystem which will be presented in the following sections is to find similar characteristics in the diet and lifestyle of users and group the ones with similar patterns. Essentially, this task can be rephrased as a pattern discovery problem in a multivariate time series. One naive solution would be to define such patterns beforehand. However, the habits and customs of people change over time. Hence, employing domain experts who can provide baseline patterns would be costly. Our system aims to derive these patterns directly from the data and to adapt them over time, when required. An example of such patterns can be observed in Figure 3.2.

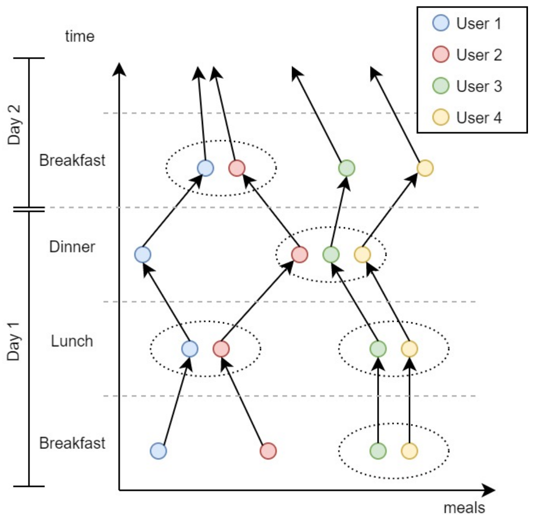


Figure 3.2: Example of user patterns

The main idea of the implementation is to make use of the identifiers of the centroids computed in the k-means phase. Instead of operating on raw data, the system assigns an integer between 1 and K to each log. Thus, the original stream of timespamped multidimensional values is converted into a univariate one. This process transforms our problem into **categorical time-series clustering**. The next logical step is to match those sequences which all follow a common pattern. We use the **Hamming distance** between the resulted univariate sequences to compute the similarity between different users. Then, the users with highest similarity are grouped together to construct candidate groups, which are then refined to extract the final groups. Lastly, for each group, we discover the most representative sequence to use as the baseline pattern of the group.

Figure 3.3 offers an overview of the implementation of the pattern matching step, alongside its interaction with the k-means pipeline. Further details about the algorithm and the implementation will be presented in the following paragraphs.

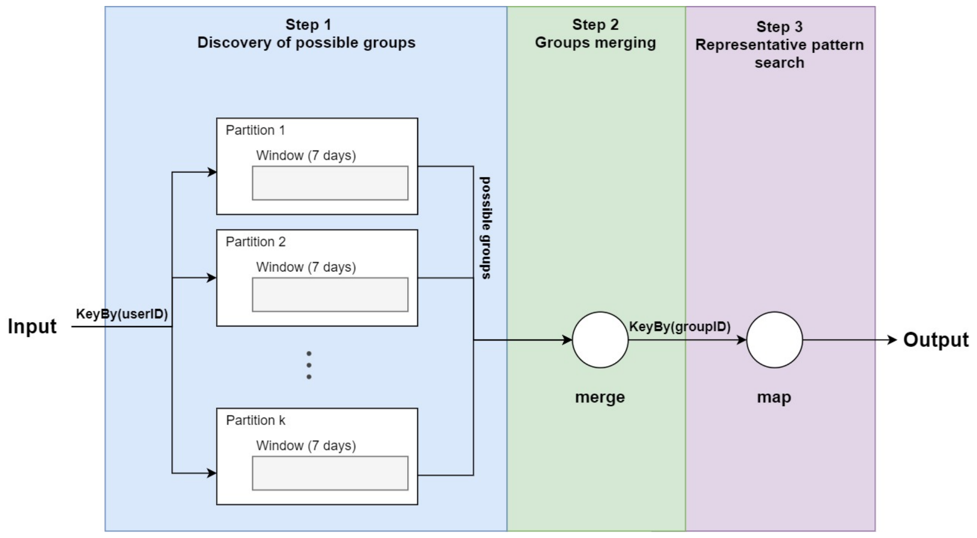


Figure 3.3: The pattern matching pipeline

3.1.2.1 Input, output, and parameters

The input and output of the pattern matching pipeline can be found in Table 3.3. As this phase is the second one in our clustering pipeline, the input is, naturally, the output of the k-means phase, with the small difference that the system now takes into consideration the timestamp and the user corresponding to each log. Thus, the input is a stream of tuples which contains a user identifier u_{id}, a timestamp *t* and an integer *c*, where *c* is the centroid identifier which was assigned by the k-means step to the log identified by user u_{id} and timestamp *t*. To maintain the parallelism, the input has to be keyed by the user identifier. The pipeline produces a list of tuples as output, which contains information about the groups that were discovered. For each group g_{id}, the pipeline outputs a list of all the users *[<users>]* that belong to that group and the most representative pattern of the group. As for the parameters, the size of the window *size*$_w$ and the similarity threshold T have to be predefined.

Table 3.3: Input, output, and parameters for the streaming pattern matching implementation

Input	stream of (u_{id}, t, c) tuples, keyed by u_{id}
Output	stream of lists of $(g_{id}, [< users >], pattern)$ tuples
Parameters	$size_w$
	T

3.1.2.2 Step 1 - Discovery of candidate groups

The initial step of the pattern matching pipeline aims to find the candidate groups of similar users. Algorithm 5 presents the flow of the operations used for this step.

Firstly, the input stream is partitioned into non-overlapping chunks measured in days (tumbling windows). Thus, for each user u_i, we assign a list D_i which contains their logs that were stored in the given time window, grouped by days. It is important to remember that the logs in each day are characterized by only one integer, which is the centroid identifier assigned previously to the meal. Thus, each day sequence will be a tuple (c_b, c_l, c_d), with $c_b, c_l, c_d \in \{1, 2, ..., K\}$ the ids of the centroids assigned to the breakfast, lunch and, respectively, dinner of that day.

Then, we compute the similarity matrix S, in which the cell $S[i, j]$ contains the similarity between the user i and the user j. This cell is computed as the Hamming distance between the sequences of user u_i and the sequences of u_j that were stored in the given time window. The method *Hamming()* takes a list containing the lists of all the logs in the recorded time window (grouped by user) and computes the similarity between all pairs of users (line 8).

Lastly, for each user u_i, we find the group of users with the most similar patterns. To do so, for each row of the similarity matrix, we add to the group the users with similarity bigger than the predefined threshold (line 13) and we output the group. Each line corresponds to one user. This aspect can be used to parallelize the computation, as each worker can receive a broadcast message of all the sequences contained in the time window. and compute a set of rows in the similarity matrix. After each step, a candidate group is produced for each user.

Algorithm 5: Pattern matching - discovery of candidate groups

Input: Stream of (u_{id}, t_i, c) tuples, keyed by u_{id}
Output: Stream of $(u_{id}, [< users >])$ tuples, keyed by u_{id}

```
1  split the input stream into **tumbling windows** of size $size_w$ days;
2  let $w$ the current window;
3  let $U = \{u_1, u_2, ..., u_n\}$ be the list of users with logs in $w$;
4  let $D = \{D_1, D_2, ..., D_N\}$ with $D_i$ the logs of user $u_i \in U$ in $w$;
5  $S = Hamming(D)$;                       // the similarity matrix
6  foreach $i \in \{1, 2, ..., n\}$ do
7      $G_i = []$ ;                        // the group of user $u_i$
8      foreach $j \in \{1, 2, ..., n\}$ do
9          if $S[i, j] > T$ then $G_i = G_i \cup \{j\}$ ;
10     end
11     output $G_i$;
12 end
```

3.1.2.3 Step 2 - Groups merging

As the grouping of users from Step 1 was performed using only information available at the level of each user by computing the Hamming distances, there might be the case that some users appear in multiple groups. This situation is not acceptable in our case, as it is required for one user to belong to only one unique group. In real life, it might be the case that the habits of one individual are somewhere between two defined groups. However, in our system, one user should be assigned to only one group, so that the choice of the federated model used to serve is unquestionable.

We choose to solve this issue by applying a straightforward mechanism: each user that appears in multiple groups after Step 1 is assigned to the group with the largest number of members. This approach is suitable for our use-case, as the average similarity of each group is not affected. Moreover, it favours the creation of well-sized groups, an aspect which will be beneficial in the training phase.

However, this approach, whilst simple and intuitive, comes with a drawback in terms of implementation, as it has to be performed by a central entity. The candidate groups have to be first gathered in a central node, which will decide the composition of the final groups.

3.1.2.4 Step 3 - Representative pattern search

The final step of the pattern matching pipeline is the identification of representative patterns for each group. This step is not closely related to the research questions proposed by this work, but can be used in scenarios where a system reconfiguration is needed. We decided to include it since it might be beneficial for future works. As mentioned before, the grouping of the users has two usages: to train separate federated models for each group and to decide which of these models to use when serving users that are new in the system or that exhibit lifestyle changes. While Step 2 is enough to fulfill the former, extra computations are needed for the latter.

It would be highly inefficient to store the patterns for all of the users and re-execute the grouping mechanism every time a new user joins. To overcome this issue, we store only the sequence of the user that proved to be the most representative for each group. For each group, we iterate over its users and decide which user is the most representative one. We define the most representative user of a group as the user that has the strongest similarity with the other members of the group. New users will, then, match their current patterns against the most representative patterns of each group and get assigned to one group. The same reasoning applies to users changing their habits. This approach is not only efficient in terms of computation, but also beneficial in terms of privacy preservation. The central coordinator stores patterns, which are essentially sequences of centroid ids, and not raw data.

3.2 Federated learning implementation

In this chapter, we will discuss the implementation details of our federated learning step. Our implementation was done using the TensorFlow Federated framework [63].

In the federated learning paradigm, the central coordinator uses updates from the clients to improve a global model. This model is then shared with all the clients, who will further try to improve it using local data. The global model is described using a set of hyperparameters. Some of the parameters will be picked accordingly for each experiment (dataset), while others will be fixed across all the experiments. The list of parameters can be found in Table 3.4.

Table 3.4: Hyperparameters for the federated model. The experiment dependent parameters will be chosen to best fit each dataset.

Hyperparameter	**Value**
Network architecture	Experiment dependent
Neurons	Experiment dependent
Batch size	Experiment dependent
Client learning rate	Experiment dependent
Server learning rate	0.05
Client optimizer	Adam
Server optimizer	Adam

As for the aggregation method, we will use the standard Federated Averaging algorithm [37]. The main idea of this algorithm is to allow the clients to perform multiple batch updates on local data and then share the updated weights with the coordinator. Then, the coordinator averages the updates received from the clients and updates the global model. It should be noted that this method requires the existence of two optimizers: one on the server level and one for the client. The former is used to apply the averaged update on the parameters of the global model, while the latter is used for the local updates.

We will study the accuracy of the federated model in two different situations: one being the embodiment of the best-case scenario and the other that showcases a real-world scenario. In the first case, we allow all the users to participate in the federated process and contribute to the centralized model. In reality, not all users are available to send their updates to the server concurrently. Hence, for our real-world study, we sample a random subset of clients who will contribute at each communication round. We chose to create a subset of 10% randomly selected clients at each round, as multiple studies show that this fraction achieves a good trade-off between model convergence and computational efficiency [37, 64, 65].

3.2.1 Integrating the clusters

In our pipeline, the federated learning step also takes the output of the clustering step as input. Thus, it must store a map which assigns the clients with their corresponding group. Given this information, the learning process proceeds as usual, but with a few tweaks to make use of the user groupings.

The procedure is simple. The server selects a sample of clients to update the global model, according to the availability of these clients. Then, instead of merging all the updates into a single global model, the server uses the information about users' groups and merges the updates of the clients into separate models, thus maintaining K federated models, where K is the number of groups.

In Figure 3.4 we depict the process in one communication round for $client_p$, but the procedure is identical for all the other clients. $client_p$ sends its updates (step 1). Then, the coordinator looks up $client_p$ and finds this client belongs to cluster $k-1$ (step 2). At step 3, the coordinator finds the model that corresponds to cluster $k-1$ ($model_{k-1}$) and updates it using the information received from $client_p$. In the final step, the coordinator shares the updated model with the client.

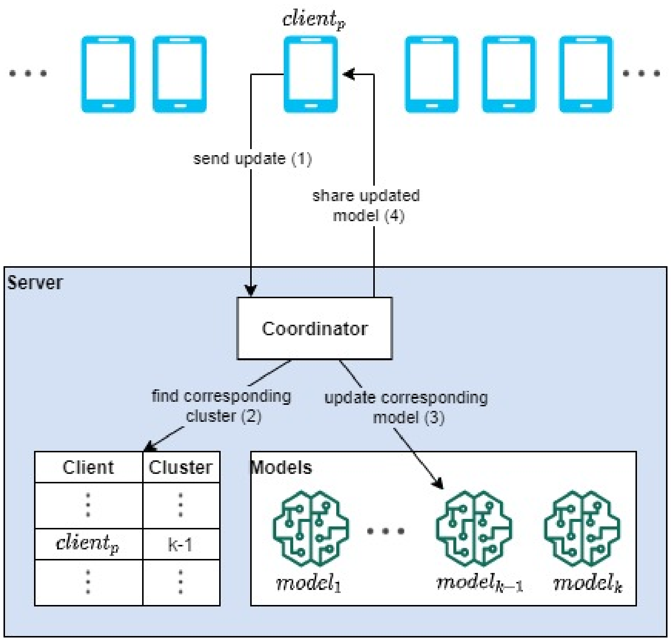
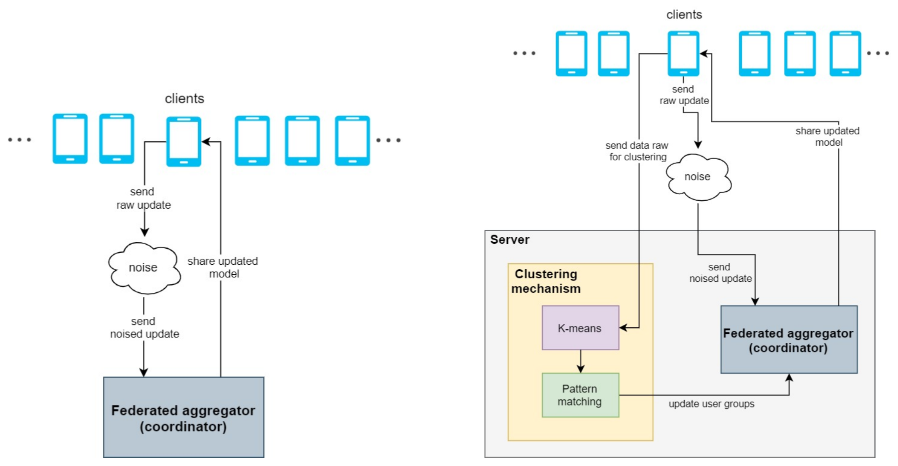


Figure 3.4: Overview of the communication round in the federated learning process with clustering. In this example we assume the clusters are grouped into k clusters

3.2.2 Client confidentiality with differential privacy

We aim to achieve client confidentiality in our system by employing differential privacy methods. In this work, we compare two privacy preservation methods. Both achieve the goal of noising the updates that are sent to the coordinator so that no sensitive information can be leaked by exchanging these parameters. For both approaches, we will vary the amount of added noise and study the impact it has on the accuracy of the system. We will perform this study on both the baseline (general) model and the clustered models.

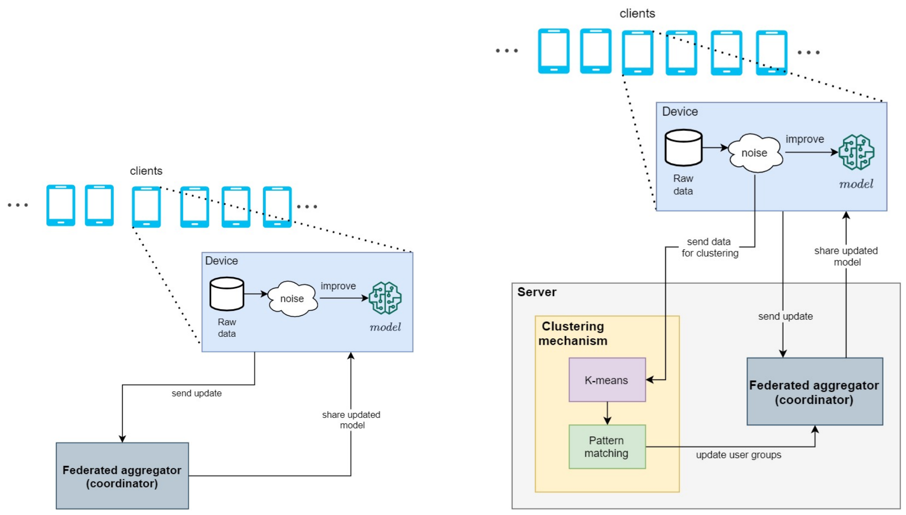
First, we noise the weights sent to the federated aggregator. Each client learns from local data and tries to improve the federated model as per the classic federated learning mechanism. The change appears when it comes to sending the update to the server. As it can be noted in Figure 3.5, instead of sending the true update, each client adds noise to this update, in both baseline and clustered scenarios. The amount of noise added to the weights directly influence both the level of privacy achieved by the system and the accuracy of the federated model. The prediction happens as usual, on each client's device and on clean data. The added noise will be sampled from the Gaussian distribution. This method has been implemented using the Tensorflow Privacy framework [66] and the choice of noise distribution has been dictated by the functionality provided by the framework. The amount of added noise is controlled by two parameters: the *clip* applied to the learned gradients, which manages the sensitivity of the optimizer to individual training points, and the *noise multiplier*, which controls the amount of noise added during training. In the end, the standard deviation of the noise is computed by multiplying the two parameters.



(a) Baseline (b) Clustered

Figure 3.5: The noisy learning scenario in a federated learning context

Although the state-of-the-art method to achieve differential privacy in a federated learning context is the noisy learning approach, Geyer, Klein, and Nabi [60] suggest that the success of this method highly depends on the size of the dataset. Because the dataset we have available might not be big enough, we test a second method, in which we add noise perturbation to the data itself. The federated learning process remains very similar to the standard process, with the small modification that the clients noise their local datasets before trying to improve the federated model. Thus, as the weights of each model are computed based on input data, the updates they send to the coordinator will implicitly contain noise. Figure 3.6 shows a higher level image of the process. It should also be noted that, in this scenario, the clustering mechanism will receive noised input, meaning that the quality of the grouping will be affected. The prediction, in this case, is performed locally and on clean data. We chose to noise the data by sampling from the Laplacian distribution because this method is popularly applied in the related literature [67, 68, 69].



(a) Baseline (b) Clustered

Figure 3.6: The noisy data scenario in a federated learning context

For both methods, we will adjust the noise added so that we can achieve a sensible level of data privacy. Tang et al. [70] examined the differential privacy methods used by Apple for macOS and iOS found that MacOS's differential privacy has an epsilon as large as 6, while for iOS the epsilon value can go up to 14. Google's version of differential privacy, named Randomized Aggregatable Privacy-Preserving Ordinal Response, claims to achieve an epsilon value of 2 in certain scenarios but can be as high as 9 [71]. However, researchers found that a value of epsilon higher than 1 does not give good privacy protection [72]. We will use these values as a guideline in our study and strive to achieve a compromise between the achieved level of privacy and the performance of the system.

Chapter 4

Evaluation methodology

The purpose of our experiments is to find an answer to our research questions. As we designed and implemented a novel multivariate time-series clustering method, it is essential to prove its effectiveness. We expect our method to be able to cluster both univariate and multivariate time series with very similar performance, as long as the distance functions are meaningful in the high-dimensional space. Thus, we choose two synthetic datasets which contain univariate time series to test our clustering method. The first dataset was chosen because it contains visible clustered time series, while the second one was chosen to show that the optimal clustering is not always obvious.

Our first research question is to determine whether we can predict the next nutritional intakes of an individual with high accuracy, in a federated context. We experiment with two datasets, one which will embody the perfect world scenario, in which no data is missing, and another that represents a real-world scenario, where the recorded data is not continuous. We first carry out an analysis to discover the best model for our configuration by running a grid search on a sample of the dataset. In our experiments we have chosen a sample of 10% of data. Then, we use this model in federated learning and assess the results in terms of the Mean Absolute Error and the Root Mean Square Error obtained by our model on the test dataset.

Then, we wish to discover whether exploiting similarities between the diets of the users can be beneficial to improve the performance of the federated model. We apply our clustering mechanism on the records of our users and group them. Then, we train separate models for each of these groups and compare the accuracies on the test datasets.

Lastly, we focus even more on the privacy aspects of our work. We are working with several health records of users that should remain private. Hence, we apply additional methods to our pipeline to preserve the privacy of the individuals. We wish to discover what is their effect on the overall accuracy. For this, we simply compare the accuracies obtained when toggling on and off these privacy-preserving approaches.

The following section will start with the characterization of our datasets and an extensive explanation of the preprocessing methods applied to our datasets of choice. Then, we describe our development process and proceed to define the evaluation methods that will quantify the success of our methods. Lastly, we mention the environmental setup of our experiments.

4.1 Data analysis and preprocessing

The following section is devoted to the process of collecting the data necessary to undertake our study. First, we focus on the synthetic datasets that will be used to assess the performance of our clustering method. Next, we describe the food, nutrition, and health diaries which will serve the purpose of answering our main research questions. We thoroughly analyse these datasets and describe all the preprocessing steps performed.

4.1.1 Synthetic datasets

We test the efficiency of our clustering method using two synthetic datasets which contain univariate time-series data: one manually generated and the Synthetic Control Chart Time Series dataset [73]. Although there is a high number of available time-series datasets, we picked these two strictly because we aim to prove the validity of our clustering method. These datasets contain clustered time series in which it is more or less trivial to assess the quality of the applied clustering algorithm.

Let us describe the manually generated dataset. The dataset contains six groups of correlated univariate time series. Each group contains 10 series and each series on one group was generated from the same sinusoidal or cosinusoidal wave with some noise. Figure 4.1 is a visual representation of the dataset. It can be clearly observed which series belong to the same cluster. We will use this dataset to test whether our clustering mechanism can identify these clusters.

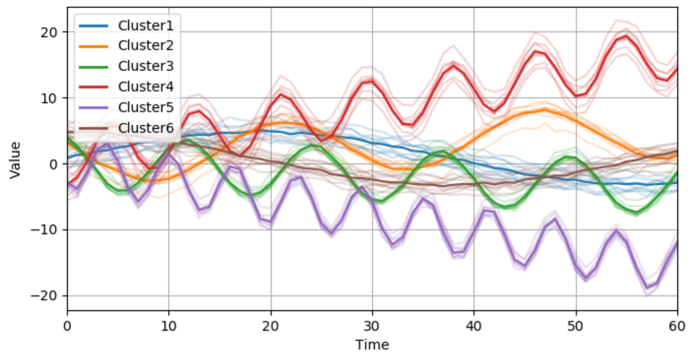


Figure 4.1: Synthetic univariate dataset with smooth repetitive oscillations

Next, we move towards a more complex dataset, the Synthetic Control Chart Time Series dataset. Similar to our manually generated one, this dataset contains 600 control charts, that were synthetically generated to fit into six clusters, based on their overall shape. Each group contains 100 time series. As it can be noted from Figure 4.2, the charts present a common central tendency but are more irregular than our manually generated dataset. This dataset shows one of the trickier aspects of time series clustering. Let us take the "Increasing trend" and "Upward shift" classes. Even though they were generated based on different tendencies it is arguable whether the two classes should be clustered together or not. The same observation applies to the "Decreasing trend" and "Downward shift classes". Thus, we believe it would be interesting to test how our clustering method behaves on this dataset.

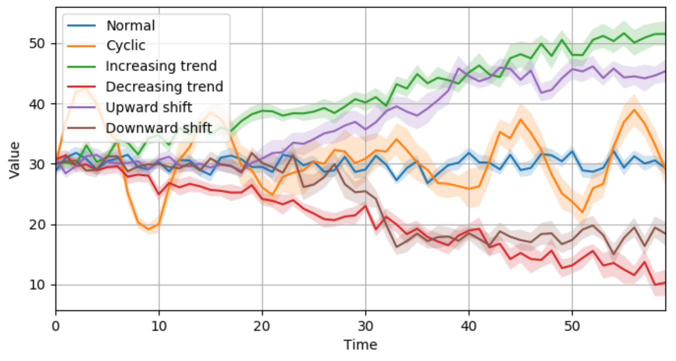


Figure 4.2: The Synthetic Control Chart Time Series dataset [73]. For visualization purposes, this figure shows a sample of 10 series from each class. Each class is depicted using a different color. For each class, an estimation of the central tendency is highlighted.

4.1.2 Diet and fitness tracking datasets

In this subsection, we describe the analysis and preprocessing performed on the two available diet and fitness tracking diaries used for our study: The MyFitnessPal dataset and the Fitbit dataset.

4.1.2.1 The MyFitnessPal dataset

The first food diary dataset we will use in our analysis is the MyFitnessPal Food Diary Dataset [74], which contains records from 9900 users who logged their foods for almost 207 days. Each entry contains an anonymized user identifier, the log date, the name of the food, and the breakdown in macronutrients: carbohydrates, protein, and fat. This dataset will be used as the best scenario for our multivariate time-series forecasting experiments.

Firstly, we analyze the number of missing values for each time series. Because we want to predict the dietary intake of each individual, the number of missing records is very important for our study. The first preprocessing performed on the dataset was to drop the series of users who recorded their meals without any label for the time of the day since there is no way of telling which food was consumed at which meal.

Next, we noted that, unsurprisingly, the subjects logged multiple foods for each meal. Thus, all the values corresponding to one meal have been aggregated into only one. Moreover, the dataset contains the following labels: breakfast, morning snack, lunch, afternoon snack, dinner, and evening snack. Although these constitute a valid time series, the records of the subject have massive variations in terms of the snack records, meaning there will be a lot of missing values in the series. We decided to limit these labels to only three, breakfast, lunch, and dinner since these are the three most important meals of a day. The snacks were, therefore, summed up with their corresponding major meal (i.e. the morning snacks were added to breakfast).

Figure 4.3 shows the missing logs for each subject after the above-mentioned preprocessing. It can be noted that the majority of individuals did not log consistently during the data collection time frame.

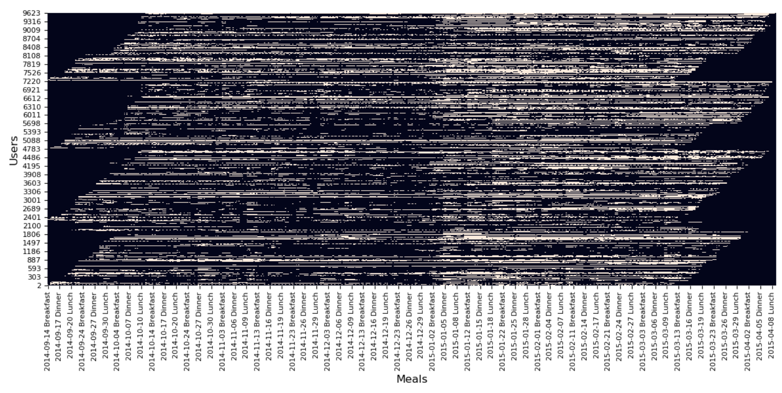


Figure 4.3: Visualization of the missing logs in the MyFitnessPal Food Diary Dataset. The black spaces represent the absence of logs for a particular meal

As the purpose of this dataset is to test the forecasting in the best scenario, we decided to pick only the users who logged concurrently, at least one meal per day and for a time period as long as possible.

Although we could have adopted methods to fill the missing dates (such as by repeating the values from the last recorded day or filling them with the average values of previous records), we noticed that the gaps are quite long (multiple consecutive days missing), which means that filling them might alter the time series too much, such that the habits of individuals are not well represented anymore, and might lead to misleading results. Thus, we chose only the users who logged every day at least one meal per day. For the missing meals of each day, we decided to treat them as meals that were simply skipped. We noticed multiple users who did not record their breakfast and the best explanation is that they are not eating anything for breakfast. Another explanation might be that they forgot to log that particular meal, but in this case, it is hard to decide how to fill that log. Thus, we just treat them as truly skipped meals.

We picked the users who logged concurrently so that we can capture the seasonality of data. We expect, for example, some users to eat differently during weekends than during weekdays and we want these users to be clustered together, as they manifest similar behaviour.

After the preprocessing step, the data set we obtained contains 89 users who recorded their meals for 151 days. This dataset will be used further for our study. We will use the nutritional breakdown for our training and prediction process. Details about the values in the dataset can be found in Table 4.1.

Table 4.1: The range of values contained in the MyFitnessPal dataset. The macronutriens represent the breakdown of the meals into carbohydrates, protein, and fat.

	Unit	**Minimum value**	**Maximum value**
Macronutrients	grams	0.5	609

4.1.2.2 The Fitbit dataset

The Fitbit dataset will be used in our study as a real-world example of health and diet tracking dataset. Considering this is the only dataset that was found available that contains sensitive data at the moment of this study, we will use it to study the effect of adding privacy preservation methods in terms of prediction accuracy. For each user, the dataset stores information such as gender, height, weight, location, body weight index (BMI) and calories burned, alongside their food records.

The Fitbit dataset contains 25 users and is being collected in our department. This dataset has been obtained from exporting the data collected by Fitbit wearable fitness trackers. We applied the same preprocessing steps that were chosen for the MyFitness pal dataset, with only one difference. After analysing the data we noticed that multiple users logged only the name and the weight of the food they consumed, but the nutritional breakdown was missing. Thus, we added another preprocessing step to this dataset, to fill the missing values. We used the Nutritionix API [75], which takes a meal name and returns its nutritional information per the given weight. Moreover, the Fitbit application is available in multiple languages. Hence, we had to translate the logs for multiple users, as the Nutritionix API only takes input in English.

It should also be noted that only 10 devices were worn at the same time, as it can be observed in Figure 4.4. Besides, the dataset is on the smaller side, which means we cannot pick only the users who logged concurrently, as we did with the MyFitnessPal dataset. Instead, we use the whole dataset.

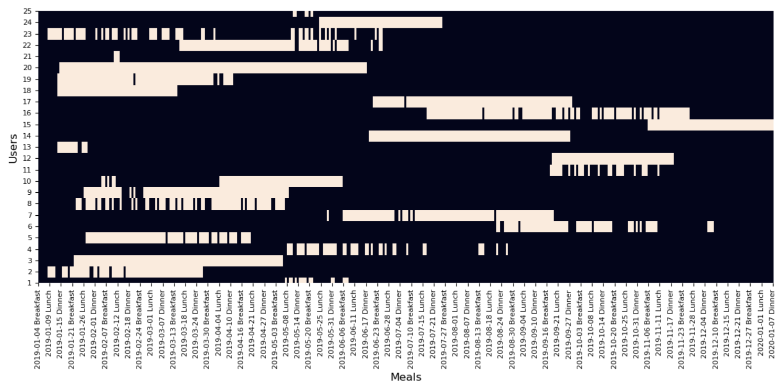


Figure 4.4: Visualization of the missing logs in the Fitbit Dataset. The black spaces represent the absence of logs for a particular meal

Most subjects did not log rigorously and that some time series present multiple gaps. However, this is usually the case with many food records, as it might not be in the interest of all individuals to record every meal. Again, we could employ methods to fill these missing values, but this might lead to a drastic change in dietary habits. Thus, we decided to drop the missing dates and re-index the time series for all the users. In this way, the first logged day of each user becomes day one and the missing timestamps are simply ignored. We expect the results on this dataset to be poorer than those obtained on the MyFitnessPal dataset considering the dietary patterns will not be perfectly captured, but we believe this dataset would bring a better understanding of how the forecasting might behave in a real-world scenario.

To test our clustering mechanism the dataset has to contain enough information so that each group can be clearly represented and learned from. Having only 25 users in a dataset would mean our model has to learn from a very small amlount of data for each resulting cluster. To overcome this issue, the dataset has been augmented using Generative adversarial networks, which resulted in a dataset of 630 users with around 2 months of logs each. This dataset has been provided to us for this work as is. Thus, the augmentation process is not part of our contributions. Both datasets will be the object of our study.

The biggest advantage of experimenting on these datasets is the private information available for each user. Even though the nutritional breakdown of the meals can lead to deriving information about the relationships between the subjects, it would be far more valuable to study how applying popular privacy preservation methods affect the results on actual sensitive data. Thus, for this dataset, we include this information and study how differential privacy affects the accuracy of our predictions.

A brief description of the features contained in this dataset can be found in Table 4.2. The macronutrients represent the breakdown of the meals of each timestamp into carbohydrates, protein, and fat. The calories burned at time t represent the number of calories the user burned between time $t-1$ and t. The same reasoning applies for active minutes, which represent the sedentary, lightly, moderately, and very active minutes of an individual. The resting heart rate is computed per day in the dataset and is repeated three times in our dataset, meaning that for the timestamps "Breakfast day n", "Lunch day n", and "Dinner day n" the resting heart rate value is the heart rate of the user computed using data from day $n-1$.

Table 4.2: The range of values contained in the Fitbit dataset. The minimum vale for macronutrients, calories burned and resting heart rate is the minimum non-zero value found in the dataset. For these features, values equal to zero in the dataset mean missing data points. For active minutes this is not the case, since a zero value would just mean lack of physical activity during that time frame.

	Unit	**Minimum value**	**Maximum value**
Macronutrients	grams	0.313	1463.108
Calories burned	kCal	467.006	2335.445
Resting heart rate	beats per minute	55.029	89.995
Active minutes	minutes	0	1242.413

4.1.3 Supervised learning from time-series data

Neural networks are designed to approximate the results of a function, given a specific input. Thus, the problem must be shaped accordingly, as pairs of inputs and labels. However, our datasets do not comply with this format and must be further preprocessed.

The data we have at hand is in the form of time series. We are going to attempt multi-step predictions, which means that we are going to forecast multiple meals, given the historical data of the user. Hannapel et al. [76] suggested in their recent study that only the memories of recently ate foods influence the next meals of individuals. Hence, it does not bring any benefit to base our predictions on long-term data. We model our data according to the Multi-Input Multi-Output (MIMO) strategy [77]. This strategy will be briefly described in the following paragraphs.

Let's assume we have a time series $[x_1, x_2, x_3, ...]$, an input sequence of size N and an output sequence of size M. In this case, we would like to predict the next M points of the series. The MIMO strategy employs learning a multiple-input multiple-output function $\hat{F}$ as it follows:

$$\hat{F}(x_{t+1}, ..., x_{t+N}) = [\hat{x}_{t+N+1}, \hat{x}_{t+N+2}, ..., \hat{x}_{t+N+M}] \tag{4.1}$$

where $\hat{x}_j$ represents the approximation of $\hat{x}_j$ through the function $\hat{F}$ and $t, j \in \{0, 1, 2, ...\}$. In other words, we predict the next M points using a sliding window of size N and slide equal to 1. It is important to note that, while the problem imposes the size of the output M, the size of the input N is picked by design. In our case, we will pick a small value for M, such as three to six meals that correspond to one to two days, due to the results discovered by Hannapel et al. [76].

4.2 Development process

The design and development stage of our project has been conducted following the literature study. Our system can be compartmentalized into three steps: one for streams clustering, one for federated learning, and one for differential privacy. For the former, there is no open-source implementation available for most of our chosen algorithms for the stream processing framework of our choice, Apache Flink [46], at the moment of writing this report. Hence, we turned again to the literature to discover the most appropriate variants of the algorithms for streams. As a result, we came up with clustering pipeline for streaming time series to satisfy our requirements. As for the federated learning process, we implemented it using TensorFlow Federated [63], an open-source framework designed for research and experimentation with federated learning. Lastly, we used the Tensorflow Privacy [66] framework for our differential privacy study.

4.3 Testing and evaluation

The problem of clustering time series is not trivial as, in many cases, it is arguable whether an optimal solution even exists and it highly depends on the problem at hand. Because we will use an intuitive method, we will prove its efficiency by reducing the complexity of the task to clustering single-value time series, as we expect our method to behave similarly in multidimensional problems. The reason behind this decision is that there is no clear consensus in the literature regarding multi-variate time series forecasting. To the best of our knowledge, the solutions proposed in the literature are either domain-dependent or not applicable to streaming data. Thus, we chose not to compare our method with other methods, but to test our algorithm against time series that can be visualized and analysed.

As for the learning phase, we assess the accuracy of our models using two metrics, the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.2}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.3}$$

where y_i represents the ground truth, $\hat{y}_i$ represents the prediction, and n is the number of observations in the test dataset.

Lastly, we have to specify how we will evaluate the performance of the models obtained from the federated learning process. Let m be the number of clients that participate in the leaning process. Let X_1, X_2, ..., X_m their local datasets. We would split the datasets as it follows: for each local dataset, we use 80% of the local data for training and 20% for testing. The overall performance is evaluated by computing the chosen errors of each local test dataset and averaging them as it follows:

$$MAE_{federated} = \frac{1}{m}\sum_{i=1}^{m} MAE_i \tag{4.4}$$

$$RMSE_{federated} = \frac{1}{m}\sum_{i=1}^{m} RMSE_i \tag{4.5}$$

It is important to remember that, when applying the clustering approach, we would be constructing separate models for each cluster. Thus, the performance of each model is evaluated similarly, using the local test sets of the users belonging to each corresponding cluster.

Lastly, we provide a minimal performance evaluation of the proposed clustering method on streaming data. Our pipeline highly relies on tumbling windows of data. Thus, we measure the execution time it takes to run each step of the mechanism on fixed-sized windows of data in a multi-threaded environment. We compare our implementation against an offline library implementation written in Python.

4.4 Experimental setup

We used Apache Flink 1.6 for the implementation of the streaming clustering mechanism. The streaming source that was used is an Apache Kafka [78] Producer written in Java 1.8 that reads the data from a file and simulates multiple clients writing to the same topic. We chose to use Kafka because of the exactly-once semantics provided by the Flink Kafka Consumer which is achieved by the Flink's checkpointing mechanism and Kafka's offset tracking.

The federated learning was implemented using TensorFlow 2.1, and TensorFlow Federated 0.13.1. We implemented the differential privacy part using the Tensorflow Privacy library.

All the experiments performed in this paper were performed on one machine with an Intel(R) Xeon(R) CPU @ 2.80GHz, 44 GB of RAM, running Ubuntu 18.10.

Chapter 5

Results

5.1 Clustering synthetic time series

Our first set of experiments aims to test the performance of our two-step clustering method. To do so, we chose two synthetic datasets that contain time-series data. In this section we analyse the results and comment on the effectiveness of our proposed method.

5.1.1 Smooth repetitive oscillations

As previously mentioned, this dataset contains six groups of correlated univariate time series and each group contains 10 series that were generated from the same sinusoidal or cosinusoidal wave with some noise.

Our clustering mechanism managed to find all six groups and clustered all 60 series correctly.

5.1.2 Control charts

The Control Charts Dataset [73] contains 600 synthetically generated control charts that are clustered into 6 groups, 100 series each.

Our clustering mechanism discovered three clusters in this dataset. It successfully identified the Normal control charts cluster and assigned all series correctly to this group. The second identified cluster is composed out of 94% of the charts with an increasing trend and 77% of the upward shift trend together. Similarly, the third identified cluster contains 96% of the charts with a decreasing trend and 73% of the downward shift trend. However, our clustering method failed to group the series with a cyclic trend. On a closer look, we discovered that, although these series were generated using a similar trend, there is a very big phase difference between the series, which makes it debatable whether they should be clustered together or not. This can be noted in Figure 5.1, where we plot a sample of the Cyclic time series.

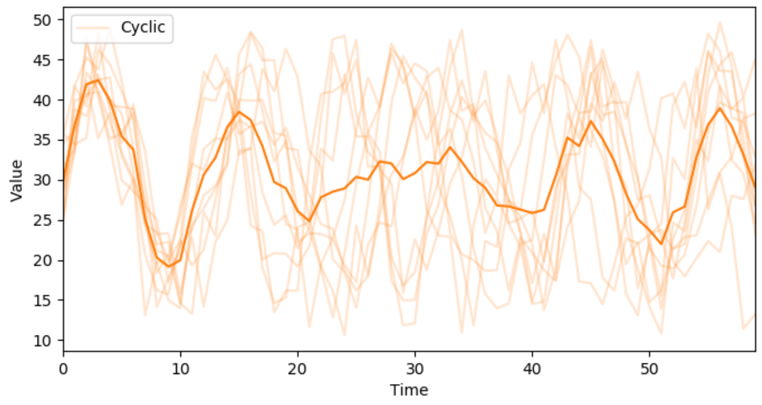


Figure 5.1: Sample of the cyclic time-series data in the Synthetic Control Charts dataset [73]

The experiments performed on this dataset showed that the algorithm is very sensitive to the chosen parameters. The window size, chosen number of clusters and the k-means initialization play a massive role in the success of the algorithm. For example, if we run the algorithm on a window size that is too small, the algorithm might fail to capture the similarities between the series. Also, choosing a value of k that is too small might result in inaccurate clusters, as the method will group time series that should not belong to the same group (such as Normal and Cyclic time series). Moreover, the problem of time-series clustering is generally difficult. As it can be noted on this dataset, if we try to group the most similar series given the time window between time 0 and time 30, it is highly debatable which series should belong to the same cluster.

This experiment concludes that the method we chose to cluster similar time series is adequate and performs well, but is very sensitive to the choice of parameters.

5.2 Case 1: MyFitnessPal dataset

In this subsection we present and analyse the results of the experiments ran on the MyFitnessPal dataset. We discuss our findings and highlight the contribution they bring in the context of our proposed research questions.

5.2.1 Choosing a model

In this subsection, we motivate the chosen hyperparameters. We aim to discover the best configuration for the dataset. We start by comparing different statistical models that are frequently used for multivariate time-series forecasting with popular neural network architectures for this task.

The comparison is presented in Table 5.1. We chose the following statistical models for our experiments: Vector Autoregression (VAR), Vector Autoregression Moving-Average (VARMA), and Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX). When it comes to neural network architectures, we experimented with Feed-Forward Neural Networks and LSTMs and GRUs. We used a sample of 10% of the dataset to perform our study.

Table 5.1: Comparison between the observed error of various statistical models against neural network architectures.

	MAE	**RMSE**
VAR	3.230	4.105
VARMA	7.160	9.051
VARMAX	3.535	4.460
FNN	0.461	0.530
LSTM	2.035	2.621
GRU	1.670	2.102

The results show that neural networks are better candidates when it comes to our problem. Thus, we decided not to move forward with the statistical models in any of our further studies.

Next, we performed a grid search for the parameters of the neural networks. In our search, we vary the following hyperparameters, aiming to discover the best configuration for our data: learning rate (η), batch size (b), and the number of neurons on each layer l. Table 5.2 shows the results of the best three models obtained, tested for different number of epochs and rounds in a federated learning process. We chose the LSTM-2 architecture with 5 epochs per round for our next experiments.

Table 5.2: Accuracy of the federated model for the best three configuration of parameters discovered using a grid search. FNN-2 refers to a Feed-Forward Neural Network with two hidden layers, both of size l. LSTM-2 and GRU-2 are defined similarly. The cell "Federated 40e - 5r" suggests that the federated model has been trained for 5 rounds, 40 epochs each.

Best models	**Parameters**	**Federated 40e - 5r**		**Federated 20e - 10r**		**Federated 10e - 20r**		**Federated 5e - 40r**	
		MAE	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**
FNN-2	$\eta = 0.001$ $b = 32$ $l = 10$	14.351	17.94	9.238	10.717	4.77	5.525	1.33	1.603
LSTM-2	$\eta = 0.001$ $b = 32$ $l = 200$	11.278	14.398	6.391	7.965	2.35	2.69	0.655	0.853
GRU-2	$\eta = 0.001$ $b = 32$ $l = 200$	10.226	12.703	16.11	18.473	4.104	5.064	1.289	1.621

5.2.2 Federated learning performance

We now move towards attempting to answer our first research questions, meaning that we wish to discover how well can we predict healthcare-related data and decide whether clustering the users brings any performance benefit.

Focusing on analysing the accuracy of the model trained on the whole dataset, we will refer to this model as the baseline model. Table 5.3 shows the performance we achieved by training the model with parameters discovered using the grid search. As it can be noted, our model can forecast the next timestamps with a Mean Absolute Error equal to 0.246, meaning that each predicted value will be at most 0.246 grams higher or lower than the ground truth. Given that the macronutrients have a minimum value of 0.5 and a maximum of 609, we believe our system is capable of highly accurate predictions.

Next, we analyse the performance of training separate models of different clusters of users. Our clustering method identified four clusters. Table 5.3 shows the achieved accuracy of each model. It is important to remember that all the models in this comparison were trained using the same hyperparameters and number of epochs. Our results on the test data show that most of the clustered models are still able to achieve high accuracy, but not as high as the baseline model. Looking at cluster 3 in particular, we notice that its model achieved the highest increase in error among the four. However, cluster 3 is the one containing the least amount of data, as it contains only 8 users. It might be the case that there is simply not enough data for the model to learn from.

Table 5.3: Comparison between the baseline model and the models trained on clusters of users for the MyFitnessPal dataset. An increase in average observed error suggests a decrease in accuracy.

		MAE	RMSE	Change in observed error	Training time (sec)
Baseline		0.246	0.319	-	5966
Clustered	**Cluster 1 (14 users)**	0.712	0.811	+178%	1082
	Cluster 2 (43 users)	0.536	0.642	+114%	3036
	Cluster 3 (8 users)	1.298	1.521	+413%	691
	Cluster 4 (10 users)	0.436	0.548	+64%	822

We should also take into consideration the training time needed for each model to achieve such high accuracy. Even though the clustered models did not outperform the baseline one, it is noteworthy that we are still able to achieve accurate predictions, but with training time drastically decreased. For example, cluster 4 manages to predict the macronutrients with a mean absolute error equal to only 0.436 after training for a period which is around 7 times smaller than the baseline model.

It would be also interesting to see how the model behaves during the learning process. Figure 5.2 shows the evolution of the Mean Absolute Error on the training dataset. It can be noted that clusters 1 and 4 seem to learn at the same pace as the baseline model. As expected, the model struggles to learn from the data on cluster 3. As for cluster 2, the model showcases a slower decrease in MAE over the training rounds. However, it should be noted that the evolution of the error is depicted using a logarithmic y-axis.

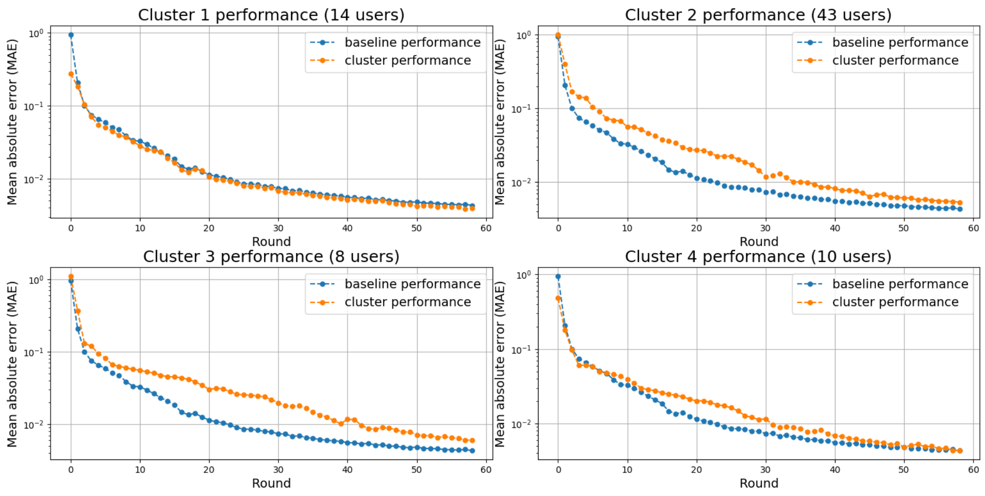


Figure 5.2: The evolution of the Mean Absolute Error of the baseline model against clustered model during the training process on the MyFitnessPal dataset.

This set of experiments lead to the following conclusions: (1) we can make very accurate predictions of users' nutrient breakdown, (2) clustering the users based on their similarities does not improve the accuracy of the prediction, but (3) we are still able to achieve high performance with less training time.

It is challenging to say why clustering the users failed to improve accuracy. One possible reason is the small amount of users available for each cluster (cluster size) for the clustering mechanism to actually exhibit its benefits. Another possible cause might be an inadequate choice of parameters in our clustering method. Since there is no information available about the dataset, a lot of assumptions had to be made based on previous studies, which might affect the performance of the method. Another plausible cause might be the homogeneity of the dataset. In this case, even if our method managed to discover some groups, the similarity inside the members of the same group might not be high enough to have the model benefit from it. A third cause might reside in the process of choosing an appropriate model for our data. As mentioned before, we ran grid search for parameters to find the ones that would perform best on our dataset. However, doing so optimizes the model for the entire dataset. Thus, using this same configuration for the clusters can be a potential source of performance drop.

5.3 Case 2-A: The original Fitbit dataset

This case study reports the results obtained on the original Fitbt dataset. This dataset serves as a real-world example o data for our specific use-case, as it can exhibit gaps in the time series. This dataset contains 25 users and more private information, such as resting heart rate and active minutes throughout the day.

Similarly to the previous case study, we perform a grid search to find the optimal hyperparameters. We include the resulted tables in the annexes since the process of discovering the parameters is the same as for the MyFitnessPal dataset.

5.3.1 Federated learning performance

Following our chosen methodology, we first investigate the performance of our federated learning procedure. Table 5.4 shows the accuracy of our baseline model against the test data. It should be noted that the number of training rounds needed to be adjusted, due to the size of the dataset. Training for a longer period showed to lead to overfitting.

We notice an accuracy drop comparing to the one achieved for the MyFitnessPal dataset. For reference, let us compare the prediction accuracy between the two datasets for the macronutrients feature. While the MyFitnessPal dataset achieved a prediction with mean absolute error equal to 0.246 (Table 5.3 for the baseline model, the same methodology on the original Fitbit dataset achieves a much higher error, of 3.27. This accuracy drop is probably caused by two factors: the considerably smaller amount of training data and the gaps in the time-series. Nonetheless, predicting the next macronutrient intake with a precision of $\pm$ 3.27 grams is still remarkable. The same reasoning applies for the other features.

Table 5.4: The prediction accuracy of the baseline model on the original Fitbit dataset.

Predicted	MAE	RMSE	Training time (sec)
Macronutrients	3.27	4.047	245
Calories burned	11.831	15.261	
Resting heart rate	0.859	1.044	
Active minutes	4.495	5.320	

Next, we cluster our users based on their similarities. Our method found 4 clusters of various sizes. We continue our study by training separate models for each of these groups. Figure 5.3 shows the evolution of the train error throughout the learning process. Three out of four models seem to have a learning curve similar to the baseline model. However, it is obvious that cluster 1 struggles to learn the patterns in its data. On a deeper analysis, we notice that, even thought cluster 1 and cluster 2 have the same number of users, cluster 1 contains the least amount of data, meaning that the logs of the users that belong to cluster 1 are very scarce.

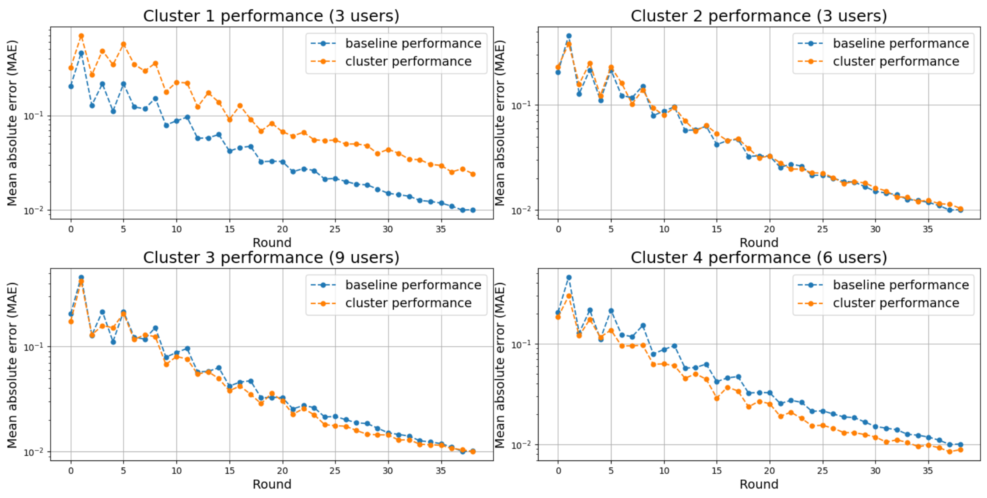


Figure 5.3: The evolution of the Mean Absolute Error of the baseline model against clustered model during the training process on the original Fitbit dataset.

Further, we evaluate the performance of our models on the test dataset. Table 5.5 contains the prediction accuracy of these models. As expected, the clusters with the least amount of available training data show a considerable decrease in accuracy (clusters 1 and 2). Groups that have more data to train on manage to achieve better predictions. Nonetheless, the baseline model outperforms all clustered models in this case.

Table 5.5: Prediction error obtained by training one model per cluster of users on the original Fitbit dataset. The average change in observed error has been recorded in comparison to the model trained on the whole dataset. An increase in average observed error suggests a decrease in accuracy.

	Predicted	**MAE**	**RMSE**	**Average change in observed error**	**Training time (sec)**
Cluster 1 (3 users)	Macronutrients	7.776	9.429	+463%	64
	Calories burned	57.544	77.070		
	Resting heart rate	9.462	12.321		
	Active minutes	19.303	23.290		
Cluster 2 (3 users)	Macronutrients	6.651	7.780	+258%	62
	Calories burned	38.816	51.437		
	Resting heart rate	5.269	6.754		
	Active minutes	12.923	15.459		
Cluster 3 (9 users)	Macronutrients	4.119	4.933	+120%	110
	Calories burned	21.791	28.054		
	Resting heart rate	2.447	3.093		
	Active minutes	7.555	9.120		
Cluster 4 (6 users)	Macronutrients	3.571	4.293	+84%	88
	Calories burned	28.165	33.643		
	Resting heart rate	2.012	2.566		
	Active minutes	7.093	8.612		

Although this dataset contains sensitive data that can be used to study the effect of applying privacy preservation methods, we choose not to proceed with it since our study shows that the amount of data in this dataset is a major impediment to drawing relevant conclusions. Hence, we focus on the augmented Fitbit datset.

5.4 Case 2-B: The augmented Fitbit dataset

This last case study reports the results obtained on the augmented Fitbt dataset. The experiments performed on this dataset aim to answer the question whether a higher amount of data can influence the outcomes we observed in our last studies.

The hyperparameters chosen for the baseline model in this section have been picked using the same methodology as in our last studies, namely from observing the results of a grid search for parameters. For more details, please consult the annexes of this study.

5.4.1 Federated learning performance

Firstly, we focus on determining how well can we predict the features of our dataset. As before, we perform multi-step forecasting and compute the

Mean Absolute Error and the Root Mean Square Error. Table 5.6 shows the performance achieved by the model trained on the whole dataset. Similarly to the MyFitnessPal case study, we conclude that our model can predict user behaviour very accurately, as our prediction is at most 0.025% as far from the ground truth value with respect to the range of each feature.

Table 5.6: The prediction accuracy of the baseline model on the augmented Fitbit dataset.

Predicted	**MAE**	**RMSE**	**Training time (sec)**
Macronutrients	0.806	1.054	9891
Calories burned	11.395	14.460	
Resting Heart Rate	0.044	0.058	
Active minutes	2.365	2.983	

Next, we cluster the users and train separate models for each group. The results show the same behaviour we encountered in the MyFitnessPal dataset, meaning that the baseline model outperforms the clustered models for all features. Because of that, these results can be found in the annexes of this paper.

This test case shows that the size of the clusters is not the only cause in the observed decline in prediction accuracy. Even if the lowest performance has been found in the model corresponding to the smallest cluster for this case study as well, we believe the amount of data in that group is enough for the model to learn (44 users). Thus, the size of the cluster is not the only cause of the low performance of the clustered models. In the following paragraphs, we explore another supposition.

The flaw in our process might be optimizing the hyperparameters for the entire dataset. The data distribution for the groups will probably be different than the one of the original dataset. We test this hypothesis by running a grid search for parameters for each cluster. We decided not to include these results since the process is identical to the results already presented. It should be mentioned that a less complex model configuration has been found appropriate for the groups, which is to be expected since the groups are much smaller than the original dataset and they contain only sequential data which exhibits similar behaviour. Moreover, it has been found that the same model configuration can cater for all groups (a small Feed-Forward Neural Network with 2 stacked layers and the same hyperparameters). This will lead to a major improvement in training time.

Figure 5.4 shows that the clustered models with optimized parameters seem to learn faster and more efficient than the baseline model. It can be noted that, after 100 rounds, each clustered model achieves a smaller training error than the baseline model.

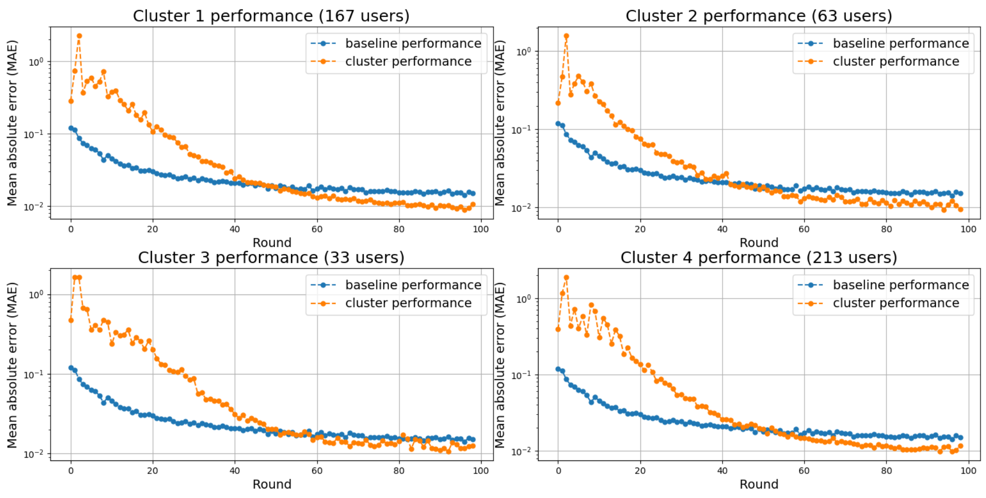


Figure 5.4: The evolution of the Mean Absolute Error of the baseline model against clustered model during the training process on the augmented Fitbit dataset with grid search performed on each group of users.

Table 5.7 shows a tremendous increase in accuracy on the test data for three out of four clusters compared to the baseline performance. Moreover, the highest training time, which has been recorded for Cluster 4, is 20 times faster than the training time of the baseline model.

As for the cluster which was outperformed by the baseline, it should be noted that the group is the smallest out of the three. We further investigated this cluster and concluded that the number of epochs and rounds used is too high for this amount of data, as we noticed that the model can achieve better test accuracy after a smaller number of training rounds. For example, after 75 rounds, the increase in mean absolute error is of only around 40%, compared with the increase of 57% we observe after 100 rounds. Thus, in the case of Cluster 3, the model is overfitting for the chosen round numbers and it might achieve better prediction accuracy if the training is stopped early. The same observation might apply for cluster 2 as well.

Table 5.7: Prediction accuracy obtained by training one model per cluster of users on the augmented Fitbit dataset. These results were obtained after running a grid search for parameters for each dataset corresponding to a cluster of users. The average change in observed error has been recorded in comparison to the model trained on the whole dataset. An increase in average observed error suggests a decrease in accuracy.

	Predicted	**MAE**	**RMSE**	**Average change in observed error**	**Training time (sec)**
Cluster 1 (167 users)	Macronutrients	0.458	0.553	-49%	374
	Calories burned	4.359	5.369		
	Resting heart rate	0.031	0.037		
	Active minutes	0.829	1.017		
Cluster 2 (63 users)	Macronutrients	0.639	0.788	-17%	199
	Calories burned	6.296	7.792		
	Resting heart rate	0.04	0.0485		
	Active minutes	2.459	2.822		
Cluster 3 (44 users)	Macronutrients	1.199	1.467	+57%	167
	Calories burned	12.152	14.65		
	Resting heart rate	0.088	0.107		
	Active minutes	4.101	4.884		
Cluster 4 (213 users)	Macronutrients	0.527	0.63	-47%	478
	Calories burned	5.493	6.8		
	Resting heart rate	0.02	0.031		
	Active minutes	1.18	1.468		

Lastly, we should address the impact of the cluster size on the overall performance of the system. It should be noted that the clusters discovered by our mechanism are imbalanced. Our results might give the impression that the size of the cluster has a great impact on the prediction accuracy and that the accuracy is directly proportional to the size of the dataset. However, this assumption is incorrect. We saw that the size of the cluster influences the results only when the neural network does not have enough data to learn from. As long as the group contains enough data, the similarity that exists between the members of each group can be leveraged to improve the prediction accuracy. We can, for example, consider the unclustered version to be a considerably larger cluster. The model trained on this group achieves poorer performance than the ones trained on real, but smaller clusters. This situation proves that the similarity between users is the only factor that determines the increase in accuracy, and not the size of the dataset.

5.4.2 Differentially private federated learning

In this final study, we focus on observing the impact of adding privacy preservation methods in the pipeline. We first investigate the performance of the system after noising the learning process. Afterwards, we ensure data privacy by noising the data itself. Lastly, we combine the two techniques to achieve higher levels or privacy preservation. As we are adding random noise to data, we ran each experiment three times and computed the average values for each configuration. The time limit constraint prevented us from running the experiments for a higher number of times. For this reason, some results might show unintuitive trends, but we strongly believe this is caused by the relatively low number of repeated experiments.

5.4.2.1 Noisy learning

We start by noising the learning process, meaning that we add noise to the updates sent to the federated aggregator. Table 5.8 presents the results of this case study. As it can be expected, adding Gaussian noise with higher standard deviation decreases the accuracy of the model, but increases the achieved privacy level.

Let us study the accuracy drop. Even for the lowest level of privacy, with a epsilon value of 10.3, the model's performance is approximately 35 times poorer than the baseline, not-noised version. This was the result of adding Gaussian noise with standard deviation equal to only 0.225.

Table 5.8: Results of noising the learning process to achieve differential privacy in the baseline scenario. The $clip$ parameter specifies the clip applied to the gradients. The $noise$ represents the amount of noise added to the gradients. sd specifies the standard deviation of the added noise. As it can be noted, the standard deviation increases from left to right, suggesting that more noise has been added. Contrary, the epsilon value decreases from left to right, suggesting that better privacy levels are achieved as we add more noise.

		$clip = 0.3$ $noise = 0.75$ $(sd = 0.225)$	$clip = 0.5$ $noise = 0.75$ $(sd = 0.375)$	$clip = 0.75$ $noise = 1.2$ $(sd = 0.9)$	$clip = 1$ $noise = 1.3$ $(sd = 1.3)$	$clip = 1.5$ $noise = 2$ $(sd = 3)$
Macronutrients	**MAE**	25.899	24.637	36.308	37.176	77.838
	RMSE	26.835	25.325	37.62	38.457	79.022
Calories burned	**MAE**	678.038	423.555	698.392	818.229	881.288
	RMSE	687.112	437.704	709.221	837.541	900.358
Resting Heart Rate	**MAE**	1.133	2.065	3.089	3.859	4.129
	RMSE	1.208	2.115	3.1581	3.939	4.194
Active minutes	**MAE**	45.942	77.553	104.254	116.546	176.347
	RMSE	49.342	80.0	107.647	121.102	180.743
Epsilon (ε)		**10.3**	**10.3**	**4.27**	**3.8**	**2.2**

When applying the noise to the clustered scenario, the results showed that the models became unable to learn anything from the data, as they showcased a mean absolute error higher that 1000 in some cases. Because of this, we decided not to include the tables with the results.

Before completing this study, we should analyse the reasons behind the drastic accuracy decrease of this scenario. Geyer, Klein, and Nabi [60] showed in their work that the number of participants in the federated averaging algorithm plays a major role on the achieved model performance. In their case, a number of 10000 clients that have 600 samples each was needed to reach accuracy similar to the non-differential private model. As our dataset is much smaller, our results are justifiable. Moreover, the fact that clustering the users caused an even greater performance drop in our experiments is also to be expected in this context, as the number of participants decreases even more when we train one model for each separate cluster.

Based on this analysis, we conclude that the noisy learning method is not appropriate for our use-case.

5.4.2.2 Noisy data

We now focus on the noisy data approach in our quest of achieving differential privacy. In this scenario, we add noise to the data itself. Each participant to the learning process learns from the noised data and tries to improve the federated model. It should also be noted that, when training on clusters of users, the clustering mechanism also receives noised data. Thus, the clusters change for each experiment.

Firstly, we will study the baseline scenario, in which there is only one federated model for all the clients. Table 5.9 showcases the performance and accuracy drop of employing various levels of data privacy. As mentioned in the Chapter 3, Google can achieve differential privacy with epsilon equal to 2 in certain conditions. Hence, this will be the starting point of our experiments. To be specific, Laplacian noise with 0 mean and $\frac{1}{\varepsilon}$ variance is added to the data. As we can see, our results show that we can reach the same level of privacy with an average increase in prediction error of only 2%. Moving further, our experiments demonstrate that we can guarantee a good trade-off between privacy and accuracy with epsilon values ranging from 1 to 0.025. We believe the best trade-off is obtained for epsilon equal to 0.1, in which case we observe an increase in prediction error of around 21%. Therefore, we study the effect of applying privacy levels equal to 1, 0.1, and 0.025 in the clustered scenario. We include only the 0.1 case, but the other two can be found in the annexes of this thesis.

We now focus on the clustering mechanism and on the impact of clustering users in our federated learning context. Firstly, as expected, clustering users with noised data alters the clusters. It can be noted that adding more noise to the data has two effects: (1) overall, less clients are assigned to clusters, and (2) the algorithm discovers a higher number of smaller clusters. As the noise

Table 5.9: Results of noising the training data to achieve differential privacy in the baseline scenario. The average increase in observed error has been recorded in comparison to the model trained without added noise. An increase in average observed error suggests a decrease in accuracy.

		$\varepsilon = 2$	$\varepsilon = 1$	$\varepsilon = 0.1$	$\varepsilon = 0.025$	$\varepsilon = 0.01$
Macronutrients	**MAE**	0.743	0.715	0.874	0.984	1.336
	RMSE	0.969	0.943	1.135	1.378	1.843
Calories burned	**MAE**	11.596	11.937	14.383	14.891	19.634
	RMSE	14.687	15.451	18.482	19.903	27.392
Resting Heart Rate	**MAE**	0.048	0.055	0.067	0.077	0.122
	RMSE	0.065	0.07	0.088	0.108	0.161
Active minutes	**MAE**	2.483	2.246	2.344	2.832	3.832
	RMSE	3.012	2.953	3.159	3.941	5.438
Average increase in observed error		**2%**	**3%**	**21%**	**37%**	**94%**

increases, the similarity between users decreases. Hence, the users that were previously very similar might still be clustered together, but the overall size of the groups is expected to decrease. The increased number of found clusters can be observed when applying privacy levels of 0.025, where the algorithm finds 5 clusters, instead of 4. The results showcasing an epsilon value of 0.025 are included in the appendix of this thesis.

We now discuss the accuracy difference obtained by clustering the users. We examine the results obtained with the best epsilon value we discovered in the baseline case, 0.1. Table 5.10 shows that, unlike the noisy learning case, clustering the users can still improve the quality of the prediction, even if the data is noised. The clusters show that the several clients maintain some degree of similarity that can be used to boost the accuracy of the model.

Table 5.10: Results of noising the training data to achieve differential privacy in the clustered scenario. The noise added achieves a privacy level with epsilon equal to 0.1. The change in observed error is computed against the baseline model in the noisy data scenario. A decrease in the average observed error suggests an increase in accuracy.

	Predicted	MAE	RMSE	Average change in observed error	Training time (sec)
Cluster 1 (73 users)	Macronutrients	0.982	1.131	-20%	222
	Calories burned	9.897	12.379		
	Resting heart rate	0.039	0.049		
	Active minutes	1.869	2.283		
Cluster 2 (37 users)	Macronutrients	1.025	1.209	-6%	150
	Calories burned	11.96	14.961		
	Resting heart rate	0.052	0.061		
	Active minutes	2.218	2.68		
Cluster 3 (161 users)	Macronutrients	0.593	0.715	-38%	392
	Calories burned	7.56	9.43		
	Resting heart rate	0.031	0.038		
	Active minutes	1.842	2.208		
Cluster 4 (97 users)	Macronutrients	0.587	0.712	-34%	262
	Calories burned	9.653	11.463		
	Resting heart rate	0.033	0.04		
	Active minutes	1.861	2.225		

Lastly, we attempt to find the reasoning behind the high accuracy obtained, even by adding a high amount of noise. Adding noise to the data has been popularly used as a regularization technique for deep neural networks to avoid overfitting and improve accuracy. However, adding too much noise has been known to decrease the performance of the model. In our case, it might be that the noising acts as too much regularization added. This could explain the very high accuracy we obtain and a relatively small performance drop compared to the non-noised model.

5.5 Performance measures of the proposed clustering pipeline

In the end, we include some brief performance measures of the clustering mechanism that was proposed in this work. We acknowledge that, given the streaming context of our application, the total execution time cannot be measured since we expect the data to be unbounded. However, all the components of our pipeline rely on the use of tumbling windows of data. Because of this, we present performance metrics measured for various windows.

The following experiments on the proposed online algorithm have parallelism equal to 8, meaning the computation is distributed among 8 different threads. The results are obtained by averaging the execution time of running the algorithm on the fixed-sized windows for each step of the pipeline. The design has been implemented in Apache Flink. For comparison, we also include the execution time obtained by an offline library implementation in Python with one execution thread.

Table 5.11 shows the results of this study. For the pattern matching step, the observed decrease in execution time is to be expected in the online version, since the execution is mainly distributed among different workers. However, we notice that the execution time of the k-means online implementation is much higher than the offline version. In this case, we are paying the price of using a Bulk Synchronous Processing model. The workers have to be synchronized after each iteration. Moreover, the workers reach consensus by exchanging messages, which also introduce a communication overhead in the system.

Table 5.11: Performance measures of the proposed clustering mechanism. The offline version is a single-threaded Python implementation. The measurements for our online versions are approximate because, at the time of conducting this study, the tumbling windows cannot be configured to use the event time in the Iterative Data Stream model of Apache Flink. Thus, the amount of data entering the window cannot be precisely controlled.

	Offline		**Online (approx.)**	
	1 day (1890 samples)	**7 days (13230 samples)**	**1 day (1890 samples)**	**7 days (13230 samples)**
Streming k-means	0.214 sec	1.398 sec	3.297 sec	21.475 sec
Pattern matching	0.366 sec	14.4 sec	0.115 sec	0.278 sec

It might be the case that the datasets we experimented with are not big enough to capture the increase in performance brought by parallelizing the computation. Given the time constraints, we do not proceed with such an analysis. However, we believe further experiments should be conducted as part of future work.

Chapter 6

Conclusion and future work

In this final chapter of our work, we summarize our findings and draw conclusions with respect to the proposed research questions. Additionally, we examine possible future extensions of this work.

6.1 Conclusion

Our thesis' work focuses on providing an online system which can forecast the dietary habits and health data of users of fitness-tracking applications and/or wearable devices, while also guaranteeing data privacy. To this extent, we have designed and implemented a pipeline capable of accurately predicting user behaviour and that can leverage similarities between individuals to improve its performance. Moreover, we achieved high levels of data protection, without compromising the accuracy of the prediction.

We designed and implemented a streaming two-phase mechanism to cluster the users based on diet and lifestyle habits. Our system is not domain-dependent and can be applied to a plenitude of other use-cases that present similarly-shaped data, as long as the problem lies in a small enough dimensional space, where it does not suffer from "the curse of dimensionality". To the best of our knowledge, there is no other open-source clustering mechanism available at the moment to process multidimensional streaming data collected from multiple inputs. The empirical study of applying this clustering method on synthetic datasets prove that the system is capable of identifying similar time series, but is very sensitive to the choice of parameters. Furthermore, the problem of time-series clustering is difficult in nature, making it difficult to discern what a good clustering looks like even for simplistic time series.

We have concluded that our federated learning system can make very accurate predictions. Our experiments show that, depending on the dataset and on the features, our predictions are no more than 0.025% far of the ground truth value with respect to the range of values. It is remarkable to note that this performance was achieved without centralizing the data, but by allowing all the clients to participate in the training process without sharing their personal data with a central coordinator.

We applied the proposed clustering system on our health-related use-case and thoroughly studies whether the discovered similarities can potentially be used as an advantage to improve the accuracy of the prediction, along with the training time. Our experiments show that the success of the clustering method is highly dependent on the data. On our rather small datasets, the models trained on clusters of users were outperformed by the baseline model, but could still achieve comparable accuracy with less training time. When moving towards a larger dataset, we observed the same trend. However, as the discovered clustered had more appropriate sizes, we discovered that using the same hyperparameters of the baseline model to train the clustered models was inappropriate.

Further, we optimized the parameters to fit each cluster by performing grid searches on all the datasets resulted by grouping and discovered that the clustered models outperform the baseline model. We discovered that a simpler model was more suitable for all the groups, which lead to tremendous improvement not only in accuracy but also in training time. For well-sized clusters, we observed a decrease in average error of more than 45%. Yet, when it comes to small groups, our experiments show that optimizing the hyperparameters is not enough. One should also consider the amount of training performed, as we noticed in our experiments that training for too many epochs leads to overfitting. Hence, we concluded that, even if the chosen grouping mechanism is appropriate, the success of the clustered federated learning method is directly influenced by the hyperparameters of the model and the chosen training amount. This shows that exploiting similarities in the data can improve the performance of a system in our use-case, but there is no such thing as a "one size fits all" method.

The data protection was guaranteed by including differential privacy into our system. As the gradients and weights sent to the federated coordinator can be maliciously manipulated to expose sensitive data, we studied two noising methods and meticulously analysed their effect on the overall performance of the system.

We first focused on a state-of-the-art method: noising the updates directly. We studied the effect of adding various levels of differential privacy preservation, which were guided by values commonly used in the industry. Our experiments showed that, even for a very low level of achieved data privacy, the accuracy of the system dropped as much as 35 times. When testing on the clustered data, we noticed that the models became unable to learn anything from the data. As other related works suggest, the number of clients participating in the learning process is crucial for the success of the noisy learning model. This explains the poor performance we saw for the baseline model and the even poorer performance observed for the clusters. Even if grouping similar data should be beneficial for the learning process, it is indisputable that, in our case, the benefits are overshadowed by the fact that an even lower number of clients participate in the learning process for each model.

Lastly, we experimented with another noising method, this time applied to the training data itself. Again, we add amounts of noise corresponding to various levels of data privacy and discover a suitable trade-off between privacy and accuracy for epsilon values ranging from 0.025 to 1. However, we believe the best trade-off we achieved has been for epsilon of 0.1, where we observed an increase in the observed mean absolute error by a factor of only 0.21.

As the literature study suggests, a value of epsilon smaller than 1 guarantees very high levels of data privacy. Additionally, we showed that this noising mechanism can benefit from clustering similar users, as we observed an improvement compared to the baseline model in the same noisy data scenario.

6.2 Future work

Firstly, our work can very easily be extended to accommodate new users joining the system. In our studies, we assumed the same users have been participating concurrently and steadily, but this is not the case in a real-world scenario. The problem of new users joining is tricky, as it would usually imply running the clustering task again to discover new groups. However, in our setup, this can be achieved by having the server store the most representative pattern of each group. This would pose no privacy concern since the pattern is essentially a sequence of univariate values that are meaningless for any malicious attacker. The most representative pattern of each group can be used to match the new user to any of the available groups, without needing to involve any other user in the process.

Secondly, our system introduces a unique opportunity for reconfiguration. As it is well-known, the federated learning paradigm implies having a random sample of users communicate with a central entity on each communication round. However, it might be the case that the model reaches convergence and there is no need for updates coming from the users to achieve high accuracy. In this case, the updates sent can pause until the system detects it should adapt based on certain metrics. This could reduce the communication necessary between the server and the clients.

We could, for example, support system reconfiguration on the client's level. One drawback in our proposed studies is that we do not cater to individuals changing their diet and lifestyle. This assumption is reasonable in our case, since the time-frame captured by our dataset is relatively small. However, our system can be adapted to account such users by monitoring the accuracy of the prediction on the device of each client. When the local system detects a drop of accuracy, it might communicate with the server to be matched to another group. In this case, the most representative pattern of each group can be used to match the client to a new group, given fresh data.

Additionally, there might be the case that overall shift in the eating patterns. For example, it has been shown that the eating habits of U.S. citizens changed drastically in the last 20 years [79]. Our k-means streaming implementation can, thus, be used as a monitoring tool for such changes and might be used to signal that the models should adapt to this change.

As it is not healthcare-related, it would be compelling to integrate our two-step clustering method for other use-cases, where appropriate. The presented performance analysis of the proposed clustering mechanism is minimal and it would be interesting to test it in under differrent loads. Lastly, Apache Flink allows jobs to run on a cluster of computers, which might make an interesting study for our pipeline.

Bibliography

[1] *Polar's 40 years of incredible firsts*. URL: https://www.polar.com/blog/40-years-of-incredible-firsts-polar-history/ (visited on 2020-08-07).

[2] *Fitbit*. URL: https://www.fitbit.com/se/home (visited on 2020-05-24).

[3] *MyFitnessPal*. URL: https://www.myfitnesspal.com/ (visited on 2020-05-24).

[4] *Number of active users of Fitbit from 2012 to 2019 (in millions)*. URL: https://www.statista.com/statistics/472600/fitbit-active-users/ (visited on 2020-08-17).

[5] Franziska Bell and Slawek Smyl. *Forecasting at Uber: An Introduction*. URL: https://eng.uber.com/forecasting-introduction/ (visited on 2020-06-08).

[6] T. Warren Liao. "Clustering of time series data - a survey". In: *Pattern Recognit.* 38.11 (2005), pp. 1857–1874. DOI: 10.1016/j.patcog.2005.01.025. URL: https://doi.org/10.1016/j.patcog.2005.01.025.

[7] Jessica Lin, Eamonn Keogh, and Wagner Truppel. "Clustering of streaming time series is meaningless". In: 2003-01, pp. 56–65. DOI: 10.1145/882082.882096.

[8] Eamonn J. Keogh and Jessica Lin. "Clustering of time-series subsequences is meaningless: implications for previous and future research". In: *Knowl. Inf. Syst.* 8.2 (2005), pp. 154–177. DOI: 10.1007/s10115-004-0172-7.

[9] Georges Hebrail and Bernard Hugueney. "Symbolic representation of long time-series". In: *Proceedings of the Applied Stochastic Models and Data Analysis Conference*. 2001, pp. 537–542.

[10] Fabian Mörchen. “Time Series Knowledge Mining”. PhD thesis. University of Marburg, 2006. Chap. 5.7, p. 179. URL: https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/pdf/pubs/2006/moerchen06tskm.

[11] Institute of Medicine. *Weighing the Options: Criteria for Evaluating Weight-Management Programs*. Ed. by Paul R. Thomas. Washington, DC: The National Academies Press, 1995. Chap. 4. ISBN: 978-0-309-05131-6. DOI: 10.17226/4756. URL: https://www.nap.edu/catalog/4756/weighing-the-options-criteria-for-evaluating-weight-management-programs.

[12] Rosa Ortega, Carmen Perez-Rodrigo, and Ana López-Sobaler. “Dietary Assessment Methods: Dietary Records”. In: *Nutricion hospitalaria* 31 (2015-02), pp. 38–45. DOI: 10.3305/nh.2015.31.sup3.8749.

[13] Stuart P. Lloyd. “Least squares quantization in PCM”. In: *IEEE Trans. Inf. Theory* 28.2 (1982), pp. 129–136. DOI: 10.1109/TIT.1982.1056489.

[14] M. R. Garey, David S. Johnson, and Hans S. Witsenhausen. “The complexity of the generalized Lloyd - Max problem”. In: *IEEE Trans. Inf. Theory* 28.2 (1982), pp. 255–256. DOI: 10.1109/TIT.1982.1056488.

[15] Inderjit S. Dhillon and Dharmendra S. Modha. “A Data-Clustering Algorithm on Distributed Memory Multiprocessors”. In: *Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD, August 15, 1999, San Diego, CA, USA, revised papers*. Ed. by Mohammed Javeed Zaki and Ching-Tien Ho. Vol. 1759. Lecture Notes in Computer Science. Springer, 1999, pp. 245–260. DOI: 10.1007/3-540-46502-2_13.

[16] Weizhong Zhao, Huifang Ma, and Qing He. “Parallel K-Means Clustering Based on MapReduce”. In: vol. 5931. 1970-01, pp. 674–679. DOI: 10.1007/978-3-642-10665-1_71.

[17] Ashish A. Golghate and Shailendra W. Shende. “Parallel K-Means Clustering Based on Hadoop and Hama”. In: 2014.

[18] Amira Soliman et al. “Decentralized and Adaptive K-Means Clustering for Non-IID Data Using HyperLogLog Counters”. In: *Advances in Knowledge Discovery and Data Mining - 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11-14, 2020, Proceedings, Part I*. Ed. by Hady W. Lauw et al. Vol. 12084. Lecture Notes in Computer Science.

Springer, 2020, pp. 343–355. DOI: 10.1007/978-3-030-47426-3_27.

[19] D. Sculley. “Web-Scale k-Means Clustering”. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW ’10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178. ISBN: 9781605587998. DOI: 10.1145/1772690.1772862.

[20] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. “Streaming k-means approximation”. In: *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*. Ed. by Yoshua Bengio et al. Curran Associates, Inc., 2009, pp. 10–18. URL: http://papers.nips.cc/paper/3812-streaming-k-means-approximation.

[21] Vladimir Braverman et al. “Streaming k-means on Well-Clusterable Data”. In: 2011-10, pp. 26–40. DOI: 10.1137/1.9781611973082.3.

[22] Marcel Ackermann et al. “StreamKM++: A Clustering Algorithms for Data Streams.” In: vol. 17. 2010-01, pp. 173–187. DOI: 10.1145/2133803.2184450.

[23] Michael Shindler, Alex Wong, and Adam Meyerson. “Fast and Accurate K-Means for Large Datasets”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. Granada, Spain: Curran Associates Inc., 2011, pp. 2375–2383. ISBN: 9781618395993.

[24] F. Garillot and G. Maas. *Stream Processing with Apache Spark: Mastering Structured Streaming and Spark Streaming*. O’Reilly Media, Incorporated, 2019. Chap. 26. ISBN: 9781491944240. URL: https://books.google.se/books?id=FqQTDQEACAAJ.

[25] Apache Spark. *Clustering - RDD-based API*. URL: http://spark.apache.org/docs/latest/mllib-clustering.html (visited on 2020-03-17).

[26] Arthur L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3 (1959), pp. 210–229.

[27] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[29] Y. Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 5 (1994-02), pp. 157–66. DOI: 10.1109/72.279181.

[30] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[31] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[32] Ian Rhys Jenkins et al. "Accident Scenario Generation with Recurrent Neural Networks". In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*. Ed. by Wei-Bin Zhang et al. IEEE, 2018, pp. 3340–3345. DOI: 10.1109/ITSC.2018.8569661.

[33] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: (2014-06). DOI: 10.3115/v1/D14-1179.

[34] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[35] Denny Britz et al. "Massive Exploration of Neural Machine Translation Architectures". In: *CoRR* abs/1703.03906 (2017). arXiv: 1703.03906. URL: http://arxiv.org/abs/1703.03906.

[36] H. Brendan McMahan et al. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[37] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. Ed. by Aarti Singh and Xiaojin (Jerry) Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1273–1282. URL: http://proceedings.mlr.press/v54/mcmahan17a.html.

[38] Brendan McMahan and Daniel Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data.* URL: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html (visited on 2020-05-17).

[39] Andrew Hard et al. "Federated Learning for Mobile Keyboard Prediction". In: *CoRR* abs/1811.03604 (2018). arXiv: 1811.03604. URL: http://arxiv.org/abs/1811.03604.

[40] Theodora Brisimi et al. "Federated learning of predictive models from federated Electronic Health Records". In: *International Journal of Medical Informatics* 112 (2018-01). DOI: 10.1016/j.ijmedinf.2018.01.007.

[41] Sumudu Samarakoon et al. "Federated Learning for Ultra-Reliable Low-Latency V2V Communications". In: *CoRR* abs/1805.09253 (2018). arXiv: 1805.09253. URL: http://arxiv.org/abs/1805.09253.

[42] Yue Zhao et al. "Federated Learning with Non-IID Data". In: *CoRR* abs/1806.00582 (2018). arXiv: 1806.00582. URL: http://arxiv.org/abs/1806.00582.

[43] Arvind Narayanan and Vitaly Shmatikov. "Robust De-anonymization of Large Sparse Datasets". In: *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*. IEEE Computer Society, 2008, pp. 111–125. DOI: 10.1109/SP.2008.33. URL: https://doi.org/10.1109/SP.2008.33.

[44] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pp. 211–407. DOI: 10.1561/0400000042. URL: https://doi.org/10.1561/0400000042.

[45] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. Springer, 2006, pp. 265–284. DOI: 10.1007/11681878_14. URL: https://doi.org/10.1007/11681878%5C_14.

[46] Ellen Friedman and Kostas Tzoumas. *Introduction to Apache Flink: Stream Processing for Real Time and Beyond.* 1st. O'Reilly Media, Inc., 2016. ISBN: 1491976586.

[47] The Apache Software Foundation. *Apache Flink — Stateful Computations over Data Streams*. URL: https://flink.apache.org/ (visited on 2020-03-17).

[48] Paris Carbone. “Scalable and Reliable Data Stream Processing”. QC 20180823. PhD thesis. KTH, Software and Computer systems, SCS, 2018, p. 180. ISBN: 978-91-7729-901-1.

[49] Leslie G. Valiant. “A Bridging Model for Parallel Computation”. In: *Commun. ACM* 33.8 (1990-08), pp. 103–111. ISSN: 0001-0782. DOI: 10.1145/79173.79181.

[50] T. Warren Liao. “A clustering procedure for exploratory mining of vector time series”. In: *Pattern Recognit.* 40.9 (2007), pp. 2550–2562. DOI: 10.1016/j.patcog.2007.01.005. URL: https://doi.org/10.1016/j.patcog.2007.01.005.

[51] Gautam Das et al. “Rule Discovery from Time Series”. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*. Ed. by Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro. AAAI Press, 1998, pp. 16–22. URL: http://www.aaai.org/Library/KDD/1998/kdd98-003.php.

[52] Mohammad Shokoohi-Yekta et al. “Discovery of Meaningful Rules in Time Series”. In: 2015-08, pp. 1085–1094. DOI: 10.1145/2783258.2783306.

[53] Lu Chen et al. “Real-time Distributed Co-Movement Pattern Detection on Streaming Trajectories”. In: *Proc. VLDB Endow.* 12.10 (2019), pp. 1208–1220. DOI: 10.14778/3339490.3339502. URL: http://www.vldb.org/pvldb/vol12/p1208-chen.pdf.

[54] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. “Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints”. In: *CoRR* abs/1910.01991 (2019). arXiv: 1910.01991. URL: http://arxiv.org/abs/1910.01991.

[55] Li Huang and Dianbo Liu. “Patient Clustering Improves Efficiency of Federated Machine Learning to predict mortality and hospital stay time using distributed Electronic Medical Records”. In: *CoRR* abs/1903.09296 (2019). arXiv: 1903.09296. URL: http://arxiv.org/abs/1903.09296.

[56] Fernando Dıaz González. “Federated Learning for Time Series Forecasting Using LSTM Networks: Exploiting Similarities Through Clustering”. MA thesis. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, 2019. URL: http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1334598&dswid=8367.

[57] Ligeng Zhu, Zhijian Liu, and Song Han. “Deep Leakage from Gradients”. In: *CoRR* abs/1906.08935 (2019). arXiv: 1906.08935. URL: http://arxiv.org/abs/1906.08935.

[58] Chuan Ma et al. “On Safeguarding Privacy and Security in the Framework of Federated Learning”. In: *CoRR* abs/1909.06512 (2019). arXiv: 1909.06512. URL: http://arxiv.org/abs/1909.06512.

[59] Reza Shokri and Vitaly Shmatikov. “Privacy-Preserving Deep Learning”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 1310–1321. DOI: 10.1145/2810103.2813687. URL: https://doi.org/10.1145/2810103.2813687.

[60] Robin C. Geyer, Tassilo Klein, and Moin Nabi. “Differentially Private Federated Learning: A Client Level Perspective”. In: *CoRR* abs/1712.07557 (2017). arXiv: 1712.07557. URL: http://arxiv.org/abs/1712.07557.

[61] Olivia Choudhury et al. “Differential Privacy-enabled Federated Learning for Sensitive Health Data”. In: *CoRR* abs/1910.02578 (2019). arXiv: 1910.02578. URL: http://arxiv.org/abs/1910.02578.

[62] Xinyi Li et al. “DP-LSTM: Differential Privacy-inspired LSTM for Stock Prediction Using Financial News”. In: *CoRR* abs/1912.10806 (2019). arXiv: 1912.10806. URL: http://arxiv.org/abs/1912.10806.

[63] *TensorFlow Federated: Machine Learning on Decentralized Data*. URL: https://www.tensorflow.org/federated (visited on 2020-06-21).

[64] Takayuki Nishio and Ryo Yonetani. “Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge”. In: *2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019*. IEEE, 2019, pp. 1–7. DOI: 10.1109/ICC.

2019.8761315. URL: https://doi.org/10.1109/ICC.2019.8761315.

[65] Yuchen Zhao et al. “Privacy-preserving activity and health monitoring on databox”. In: *Proceedings of the 3rd International Workshop on Edge Systems, Analytics and Networking, EdgeSys@EuroSys 2020, Heraklion, Greece, April 27, 2020*. Ed. by Aaron Yi Ding and Richard Mortier. ACM, 2020, pp. 49–54. DOI: 10.1145/3378679.3394529. URL: https://doi.org/10.1145/3378679.3394529.

[66] *TensorFlow Federated*. URL: https://github.com/tensorflow/privacy (visited on 2020-07-25).

[67] Yonghui Xiao and Li Xiong. “Protecting Locations with Differential Privacy under Temporal Correlations”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 1298–1309. DOI: 10.1145/2810103.2813640. URL: https://doi.org/10.1145/2810103.2813640.

[68] Yang Cao et al. “Quantifying Differential Privacy in Continuous Data Release Under Temporal Correlations”. In: *IEEE Trans. Knowl. Data Eng.* 31.7 (2019), pp. 1281–1295. DOI: 10.1109/TKDE.2018.2824328. URL: https://doi.org/10.1109/TKDE.2018.2824328.

[69] Stacey Truex et al. “A Hybrid Approach to Privacy-Preserving Federated Learning - (Extended Abstract)”. In: *Informatik Spektrum* 42.5 (2019), pp. 356–357. DOI: 10.1007/s00287-019-01205-x. URL: https://doi.org/10.1007/s00287-019-01205-x.

[70] Jun Tang et al. “Privacy Loss in Apple’s Implementation of Differential Privacy on MacOS 10.12”. In: *CoRR* abs/1709.02753 (2017). arXiv: 1709.02753. URL: http://arxiv.org/abs/1709.02753.

[71] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM, 2014, pp. 1054–1067. DOI: 10.1145/2660267.2660348. URL: https://doi.org/10.1145/2660267.2660348.

[72] *How One of Apple's Key Privacy Safeguards Falls Short.* URL: `https://aircloak.com/history-of-data-anonymization/` (visited on 2020-08-08).

[73] *Synthetic Control Chart Time Series Data Set.* Date donated: 08/06/1999. URL: `https://archive.ics.uci.edu/ml/datasets/synthetic+control+chart+time+series` (visited on 2020-05-22).

[74] Ingmar Weber and Palakorn Achananuparp. "Insights from Machine-Learned Diet Success Prediction". In: *Biocomputing 2016: Proceedings of the Pacific Symposium, Kohala Coast, Hawaii, USA, January 4-8, 2016.* Ed. by Russ B. Altman et al. 2016, pp. 540–551. URL: `http://psb.stanford.edu/psb-online/proceedings/psb16/weber.pdf`.

[75] Nutritionx. *Nutritionx API.* URL: `https://developer.nutritionix.com/` (visited on 2020-08-02).

[76] Reilly Hannapel et al. "Postmeal Optogenetic Inhibition of Dorsal or Ventral Hippocampal Pyramidal Neurons Increases Future Intake". In: *eneuro* 6 (2019-01), ENEURO.0457–18.2018. DOI: `10.1523/ENEURO.0457-18.2018`.

[77] Gianluca Bontempi, Mauro Birattari, and Hugues Bersini. "Local Learning for Iterated Time-Series Prediction". In: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999.* Ed. by Ivan Bratko and Saso Dzeroski. Morgan Kaufmann, 1999, pp. 32–38.

[78] *Apache Kafka. A distributed streaming platform.* URL: `https://kafka.apache.org/` (visited on 2020-05-27).

[79] *The New Food Fights: U.S. Public Divides Over Food Science.* URL: `https://www.pewresearch.org/science/2016/12/01/the-new-food-fights/` (visited on 2020-08-09).

Appendix A

Additional grid search results

A.1 The original Fitbit dataset

Table A.1: Accuracy of the federated model for the best three configuration of parameters discovered using a grid search on a sample of the original Fitbit dataset. FNN-2 refers to a Feed-Forward Neural Network with two hidden layers, both of size l. LSTM-2 and GRU-2 are defined similarly. The cell "Federated 40e - 5r" suggests that the federated model has been trained for 5 rounds, 40 epochs each.

Best models	**Parameters**	**Predicted**	**Federated 40e - 5r**		**Federated 20e -10r**		**Federated 10e - 20r**		**Federated 5e - 40r**	
			MAE	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**
	$\eta = 0.001$	Macronutrients	23.944	28.857	15.464	19.944	7.889	9.275	1.872	2.314
FNN-2	$b = 32$	Calories burned	161.001	191.585	375.428	384.657	67.801	73.705	12.453	15.194
	$l = 50$	Resting Heart Rate	7.919	9.073	10.131	12.183	11.316	11.566	1.380	1.529
	$\eta = 0.01$	Macronutrients	60.068	77.540	12.443	16.535	6.8261	10.439	6.948	10.215
LSTM-2	$b = 32$	Calories burned	969.657	1190.138	72.283	95.205	35.461	53.183	27.503	36.121
	$l = 200$	Resting Heart Rate	20.991	22.371	5.399	7.266	4.459	6.670	2.940	5.666
	$\eta = 0.001$	Macronutrients	10.824	14.014	8.966	10.902	3.954	5.191	3.008	4.372
GRU-2	$b = 32$	Calories burned	108.388	146.420	46.078	61.324	55.956	64.596	30.340	40.347
	$l = 200$	Resting Heart Rate	9.301	12.216	3.681	4.508	1.359	1.749	1.479	2.338

A.2 The augmented Fitbit dataset

Table A.2: Accuracy of the federated model for the best three configuration of parameters discovered using a grid search on a sample of the augmented Fitbit dataset. FNN-2 refers to a Feed-Forward Neural Network with two hidden layers, both of size l. LSTM-2 and GRU-2 are defined similarly. The cell "Federated 40e - 5r" suggests that the federated model has been trained for 5 rounds, 40 epochs each.

Best models	**Parameters**	**Predicted**	**Federated 40e - 5r**		**Federated 20e - 10r**		**Federated 10e - 20r**		**Federated 5e - 40r**	
			MAE	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**
FNN-2	$\eta = 0.001$ $b = 32$ $l = 200$	Macronutrients	17.624	19.299	10.585	12.111	12.757	14.179	13.904	15.297
		Calories burned	141.152	170.450	112.453	135.17	129.493	153.200	114.327	138.149
		Resting Heart Rate	1.106	1.264	1.04	1.186	0.497	0.622	0.925	1.054
		Active minutes	32.270	37.948	20.211	24.249	19.650	23.401	26.616	31.388
LSTM-2	$\eta = 0.01$ $b = 32$ $l = 200$	Macronutrients	9.531	11.105	8.759	10.088	9.189	10.524	9.168	10.739
		Calories burned	109.456	135.142	97.573	119.399	95.849	117.053	110.372	136.5111
		Resting Heart Rate	0.691	0.807	0.631	0.734	0.647	0.750	0.648	0.759
		Active minutes	22.596	27.248	21.133	25.425	21.883	26.226	21.632	26.165
GRU-2	$\eta = 0.01$ $b = 32$ $l = 100$	Macronutrients	11.005	12.518	10.529	11.761	11.651	12.905	9.882	11.268
		Calories burned	110.227	134.322	102.101	124.151	100.859	122.524	100.227	122.354
		Resting Heart Rate	0.782	0.902	0.714	0.821	0.772	0.883	0.702	0.811
		Active minutes	25.121	29.788	21.984	26.337	23.364	27.834	22.977	27.358

Appendix B

Additional clustered federated learning results

B.1 The augmented Fibit dataset

B.1.1 Preliminary results

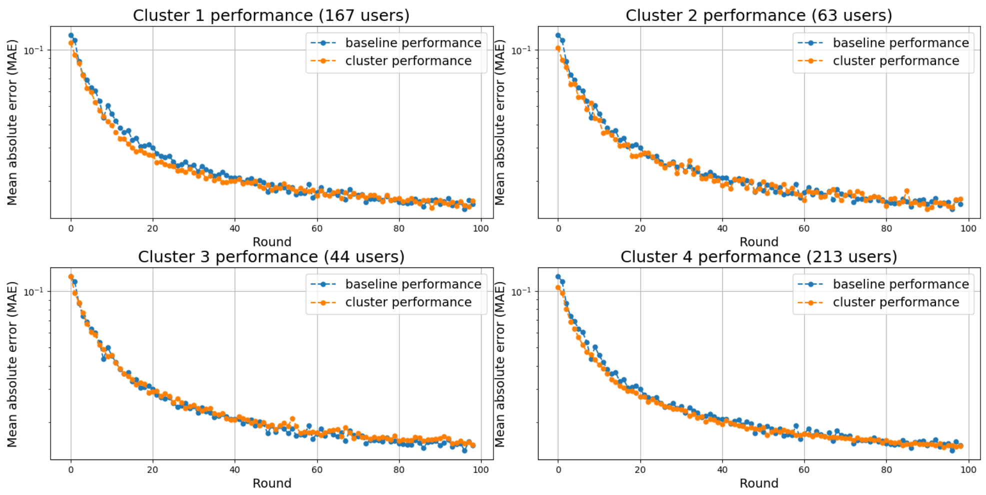


Figure B.1: The evolution of the Mean Absolute Error of the baseline model against clustered model during the training process on the augmented Fitbit dataset without grid search performed on each group of users.

Table B.1: Accuracy obtained by training one model per cluster of users on the augmented Fitbit dataset. These results were obtained before running a grid search for parameters for each dataset corresponding to a cluster of users. The average change in observed error has been recorded in comparison to the model trained on the whole dataset. An increase in average observed error suggests a decrease in accuracy.

	Predicted	**MAE**	**RMSE**	**Average change in observed error**	**Training time (sec)**
Cluster 1 (167 users)	Macronutrients	1.001	1.277	+24%	2761
	Calories burned	14.248	18.451		
	Resting heart rate	0.063	0.079		
	Active minutes	2.476	3.176		
Cluster 2 (63 users)	Macronutrients	1.412	1.798	+50%	1219
	Calories burned	12.015	15.370		
	Resting heart rate	0.081	0.105		
	Active minutes	3.269	4.058		
Cluster 3 (44 users)	Macronutrients	1.771	2.289	+124%	1020
	Calories burned	20.327	27.603		
	Resting heart rate	0.136	0.179		
	Active minutes	4.498	5.926		
Cluster 4 (213 users)	Macronutrients	1.036	1.370	+25%	3531
	Calories burned	12.683	16.216		
	Resting heart rate	0.053	0.071		
	Active minutes	3.374	4.180		

B.1.2 Differential privacy - noisy data approach

Table B.2: Results of noising the training data to achieve differential privacy in the clustered scenario. The noise added achieves a privacy level with epsilon equal to 0.025. The change in observed error is computed against the baseline model in the noisy data scenario. A decrease in the average observed error suggests an increase in accuracy

	Predicted	**MAE**	**RMSE**	**Average change in observed error**	**Training time (sec)**
Cluster 1 (112 users)	Macronutrients	0.783	0.888	-38%	294
	Calories burned	4.784	5.7		
	Resting heart rate	0.052	0.063		
	Active minutes	1.875	2.103		
Cluster 2 (57users)	Macronutrients	0.643	0.745	-40%	187
	Calories burned	6.125	7.317		
	Resting heart rate	0.054	0.064		
	Active minutes	1.79	2.067		
Cluster 3 (78 users)	Macronutrients	0.632	0.736	-42%	226
	Calories burned	6.206	7.525		
	Resting heart rate	0.049	0.058		
	Active minutes	1.668	1.973		
Cluster 4 (35 users)	Macronutrients	0.636	0.745	-36%	149
	Calories burned	8.957	10.64		
	Resting heart rate	0.052	0.062		
	Active minutes	1.769	2.093		
Cluster 5 (54 users)	Macronutrients	0.639	0.75	-34%	192
	Calories burned	10.76	12.491		
	Resting heart rate	0.049	0.059		
	Active minutes	1.734	2.059		

Table B.3: Results of noising the training data to achieve differential privacy in the clustered scenario. The noise added achieves a privacy level with epsilon equal to 1. The change in observed error is computed against the baseline model in the noisy data scenario. A decrease in the average observed error suggests an increase in accuracy.

	Predicted	**MAE**	**RMSE**	**Average change in observed error**	**Training time (sec)**
Cluster 1 (37 users)	Macronutrients	0.692	0.863	+12%	148
	Calories burned	12.359	14.447		
	Resting heart rate	0.044	0.055		
	Active minutes	3.791	4.121		
Cluster 2 (167 users)	Macronutrients	0.497	0.597	-20%	385
	Calories burned	8.687	10.393		
	Resting heart rate	0.041	0.048		
	Active minutes	1.751	2.051		
Cluster 3 (47 users)	Macronutrients	0.538	0.649	-22%	169
	Calories burned	8.752	10.598		
	Resting heart rate	0.041	0.049		
	Active minutes	1.928	2.263		
Cluster 4 (108 users)	Macronutrients	0.571	0.679	-22%	279
	Calories burned	9.235	11.176		
	Resting heart rate	0.042	0.05		
	Active minutes	1.723	2.038		

TRITA-EECS-EX-2020:615